RAZVOJ PROGRAMSKOG RJEŠENJA ZA UPRAVLJANJE MALIM DOGAĐAJIMA

Kelentrić, Josip

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

RAZVOJ PROGRAMSKOG RJEŠENJA ZA UPRAVLJANJE MALIM DOGAĐAJIMA

Josip Kelentrić

Zagreb, veljača 2023.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 28. 02. 2023.

Predgovor

Zahvaljujem mentoru, profesoru Danielu Beleu, koji je iskazao povjerenje te svojim znanjem i sugestijama doprinio nastanku ovog rada. Želim se zahvaliti i svim profesorima, asistentima i djelatnicima Visokog učilišta Algebra koji su svoje znanje prenijeli na mene. Također, zahvaljujem se roditeljima, prijateljima i kolegama koji su me podupirali i motivirali tijekom studija.

A Gradišćanska 24, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Josip Kelentrić, JMBAG 0321012157, OIB 42188833497 u šk. godini 2021./2022., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 30.11.2021. godine, ima odobrenu izradu završnog rada

s temom: **Razvoj programskog rješenja za upravljanje malim događajima**

i sažetkom rada: Student će svojim radom prikazati napredno korištenje inovativnih tehnologija u kontekstu razvoja sustava za upravljanje malim događajima. Rezultat ovog završnog rada je sustav koji omogućuje interaktvino i napredno koristenje tehnologije u svrhu upravljanja i promoviranja malih događaja.

Mentor je: Daniel Bele.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

Potpis studenta:	Potpis mentora:	Potpis predsjednika povjerenstva:

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

Sažetak

Manji događaji poput opuštenih druženja ili zajedničkih sudjelovanja u hobijima sastavni su dio društvenog života ljudi. Ovaj završni rad bavi se izradom programskog rješenja za upravljanje manjim događajima. Sama ideja rada nastala je zbog nedostatka takvog sustava na tržištu. Sustav se sastoji od tri sloja, a unutar rada opisana je izrada i funkcionalnost podatkovnog, poslužiteljskog i klijentskog sloja koji u cjelini čine navedeni sustav. Rezultat rada je programsko rješenje u obliku Android mobilne aplikacije kojoj su glavni ciljevi pretraživanje i promoviranje manjih događaja te prikupljanje sudionika sa zajedničkim interesom za neki događaj.

Ključne riječi: događaji, Android, mobilna aplikacija, pretraživanje, promoviranje

Abstract

Smaller events such as casual get-togethers or joint participation in hobbies are an integral part of people's social life. This final thesis deals with the creation of a software solution for managing small events. The very idea of this thesis arose due to the lack of such a system on the market. The system consists of three layers, and the paper describes the creation and functionality of the dana, server and client layers that make up the system as a whole. The result of the paper is a software solution in the form of an Android mobile application whose main goals are to find and promote smaller events and to gather participants with a common interest in an event.

Ključne riječi: events, Android, mobile application, find, promote

Sadržaj

1. Uvod 3

2. Problematika vezana uz upravljanje manjim događajima i prijedlog rješenja 4

2.1. Ograničenja trenutnih rješenja 4

2.2. Prijedlog rješenja 5

2.3. Usporedba s postojećim rješenjima 6

2.4. Istraživanje o potrebnosti sustava za manje događaje 7

2.4.1. Provođenje ankete 7

2.4.2. Rezultati ankete 7

3. Arhitektura sustava 12

3.1. Skica i opis arhitekture sustava 12

3.2. Korištene tehnologije i jezici 13

3.2.1. Podatkovni sloj 13

3.2.2. Poslužiteljski sloj 13

3.2.3. Klijentski sloj 16

4. Implementacija sustava 18

4.1. Podatkovni sloj 18

4.2. Poslužiteljski sloj 18

4.2.1. Pristupne točke 18

4.2.2. Dizajn API-ja 19

4.3. Klijentski sloj 25

4.3.1. Korisničko sučelje 26

4.3.2. Opis funkcionalnosti aplikacije 29

5. Testiranje i analiza gotovog programskoga rješenja 39

Zaključak 42

Popis kratica 44

Popis slika 45

Popis kôdova 46

Literatura 47

1. Uvod

Društveni događaji sastavni su dio društvenog života ljudi, a kvalitetan društveni život uvelike povećava kvalitetu života općenito. Ovo je programsko rješenje zamišljeno kao sustav koji olakšava promoviranje i pronalazak manjih društvenih događaja. Sustav omogućava korisnicima da sa svim ostalim korisnicima podijele događaj koji organiziraju i sve njegove detalje. Također, omogućava i napredno pretraživanje postojećih događaja prema vrsti i mjestu događanja.

Trenutni način promocije i pretraživanja događaja uglavnom se odvija putem društvenih mreža te na tržištu trenutno ne postoji niti jedan sustav koji se isključivo bavi događajima manjih opsega. U sljedećem poglavlju analiziraju se rješenja koja su trenutno dostupna na tržištu i predlaže se rješenje nedostatka postojećih rješenja. Kako bi se ispitala potreba za predloženim rješenjem, provodi se istraživanje u obliku ankete.

Arhitektura sustava je troslojna, a sastoji se od podatkovnog, poslužiteljskog i klijentskog sloja. U trećem poglavlju opisuju se svrhe i uloge pojedinačnih slojeva sustava. Uz to, navode se i opisuju sve tehnologije i programski jezici korišteni za realizaciju navedenih slojeva. Programski kod pisan je engleskim jezikom i poštuje sva načela objektno orijentiranog programiranja.

Četvrto poglavlje bavi se implementacijom sustava, gdje se za svaki sloj detaljno opisuje njegov način implementiranja.

U zadnjem poglavlju provodi se testiranje i analiza gotovoga programskog rješenja. Kroz različite načine testiranja osigurava se da sve implementirane funkcionalnosti programskog rješenja rade na zamišljen način. Za kraj, analizira se što je postignuto gotovim rješenjem i koji su njegove prednosti i nedostaci. S obzirom na to da u rješenju ima prostora za napredak, navodi se i nekoliko prijedloga funkcionalnosti koje bi nadogradile sustav u budućnosti.

2. Problematika vezana uz upravljanje manjim događajima i prijedlog rješenja

Za pregled i objavu manjih događaja, kao što su mali koncerti, izlasci s prijateljima, umjetničke izložbe ili okupljanja, trenutno ne postoji sustav koji je posvećen isključivo njima. Trenutna rješenja, koja ljudi koriste kako bi pronašli takve događaje, društvene su mreže, gdje se uz sav njihov sadržaj mogu pronaći i događaji. Uz društvene mreže, događaji se mogu pronaći i promovirati putem foruma, konvencionalnih medija, internet portala i blogova te usmenom predajom.

2.1. Ograničenja trenutnih rješenja

Društvene mreže, kao što su Facebook i Instagram, u današnje vrijeme se uvelike koriste kako bi se promovirali događaji bilo koje veličine. Korisnici tih mreža vrlo jednostavno mogu napraviti objavu o događaju koji organiziraju, a koju će vidjeti iznimno velik broj ljudi diljem svijeta. Ipak, u moru sadržaja jedne društvene mreže, kao što je Facebook koji koristi 2,96 milijardi ljudi na svijetu, teško je pronaći neki događaj ako korisnik ne zna što točno traži. [1] Iako Facebook ima zaseban dio svoje aplikacije fokusiran na događaje, tamo se uglavnom mogu pronaći veći događaji poput koncerata, festivala, sajmova i sportskih natjecanja.

Osim Facebooka, većina događaja se promovira i putem Instagram društvene mreže koja je sadržajno vrlo slična Facebooku. Veliko ograničenje Instagrama je način objave sadržaja, gdje je moguće kreirati objavu samo u obliku slike s nekim kratkim opisom, što je nedovoljno za objavu događaja. Uz to, potrebno je pratiti nekog korisnika kako bi se mogle vidjeti njegove objave, što uvelike ograničava količinu ljudi koje će ta objava doseći.

Internetski forumi također se mogu koristiti za objavu manjih događaja kako bi se privukao što veći broj ljudi sa sličnim interesima, ali nedostaci s kojima se korisnici mogu susresti su nestrukturiranost sadržaja i mali broj korisnika koje će objava doseći.

Uz internetske medije, tu su i konvencionalni mediji putem kojih je moguće saznati za događaje. Ograničenje konvencionalnih medija, poput televizije ili novina, jest što se uglavnom veći događaji promoviraju putem njih, a to znači da je teško pronaći manje

događaje. Uz to, promoviranje događaja putem konvencionalnih medija nije besplatno, i to organizatorima manjih događaja u većini slučajeva nije pogodno.

Čest način promoviranja, ali i pronalaska manjih događaja je usmena predaja, gdje organizator može kontaktirati ljude kako bi ih obavijestio o događaju koji organizira. Ograničenje usmene predaje je količina vremena koju javljanje pojedinačnoj osobi koju bi događaj zanimao oduzima. Uz vremensko ograničenje, tu je i ograničena količina ljudi do koju će se ovim putem dosegnuti.

Najveće ograničenje postojećih rješenja je njihova količina zbog koje netko tko želi pronaći ili promovirati događaj mora koristiti više različitih aplikacija, alata i izvora.

2.2. Prijedlog rješenja

Prijedlog za rješenje je izrada sustava, u obliku mobilne aplikacije, koji će omogućiti brzo i jednostavno pretraživanje te objavljivanje manjih događaja.

Pretraga događaja odvijat će se prema lokaciji putem karte, s fokusom na lokaciju gdje se korisnik trenutno nalazi. Događaje je moguće i filtrirati po njihovom tipu tako da se na karti prikazuju događaji samo onog tipa koji korisnik želi.

Bilo koji korisnik aplikacije može organizirati i objaviti svoj događaj, pri čemu se unose sve informacije koje bi mogle biti korisne potencijalnom sudioniku. Te su informacije datum i vrijeme, mjesto, vrsta i detaljniji opis. Događaju je moguće postaviti i cijenu prisustvovanja ako organizator želi naplaćivati svoj događaj.

Organizatori također mogu postaviti događaj kao privatni, što znači da ako neki korisnik želi prisustvovati događaju, mora biti prihvaćen od strane organizatora.

Nakon što je neki događaj završio, moguće je dodati recenziju za događaj s ciljem da se ostalim korisnicima olakša odabir događaja u kojem žele sudjelovati.

Korisnicima na raspolaganju stoji i funkcionalnost razgovora (engl. *chat*), gdje je moguće kontaktirati organizatora događaja u slučaju nekih pitanja.

Ovim rješenjem korisnicima se predstavlja sustav koji na jednome mjestu sadrži sve potrebno za pronalazak i promociju događaja.

2.3. Usporedba s postojećim rješenjima

Jedino postojeće rješenje, koje je donekle slično predloženom rješenju, jest Facebook, odnosno dio Facebooka koji služi za događaje. Za prikaz događaja na Facebooku koristi se popis događaja koji sadrži sve trenutno dostupne događaje na temelju korisnikovih preferencija, a koje se spremaju tijekom korištenja Facebooka.

U odnosu na Facebook, predloženo rješenje koristi kartu za prikaz događaja kako bi se korisnicima olakšao pregled događaja koji su u njihovoj blizini, ali i da vrlo lako mogu pretražiti i događaje u drugim gradovima.

Kako bi se popis događaja filtrirao, Facebook nudi devet različitih filtra, a to su:

- Najpopularnije – prikaz trenutno najpopularnijih događaja
- Lokalno – prikaz događaja koji se odvijaju blizu korisnikove lokacije
- Ovaj tjedan – događaji koji započinju ovaj tjedan
- Prijatelji – događaji za koje su korisnikovi Facebook prijatelji izrazili interes
- Grupe – događaji organizirani od strane grupa u kojima je korisnik član
- Na internetu – internetski događaji, odnosno događaji koji se odvijaju putem interneta
- Pratim – događaji koje je korisnik odlučio pratiti
- Predavanja
- Noćni život.

Predloženo rješenje također nudi filtriranje događaja, ali isključivo prema tipu događaja. Prilikom postavljanja filtra, na karti se prikazuju isključivo događaji koji odgovaraju odabranom tipu.

Klikom na neki događaj iz popisa, Facebook otvara novi prozor u kojem se prikazuju informacije o odabranom događaju, kao što su vrijeme i mjesto, organizator i opis. Ova funkcionalnost dostupna je i u predloženom rješenju, gdje se informacije o događaju prikazuju na sličan način.

Funkcionalnost koju Facebook sadrži, a koja nije implementirana u predloženom rješenju, jest označavanje interesa za neki događaj sa „Zanima me“ ili „Dolazim“. Ova funkcionalnost

je vrlo korisna iz razloga što drugim korisnicima prikazuje koliko osoba će prisustvovati događaju.

Recenziranje nekog događaja nakon njegova završetka na Facebooku nije dostupno, a to znači da korisnici nemaju mjerilo koje prikazuje kvalitetu organizatora i događaja koje je organizirao. Iz tog razloga, predloženo rješenje implementira takvu funkcionalnost kako bi korisnici mogli lakše odlučiti žele li prisustvovati nekom događaju.

U oba rješenja moguće je i kontaktirati organizatora u slučaju pitanja od strane korisnika.

2.4. Istraživanje o potrebnosti sustava za manje događaje

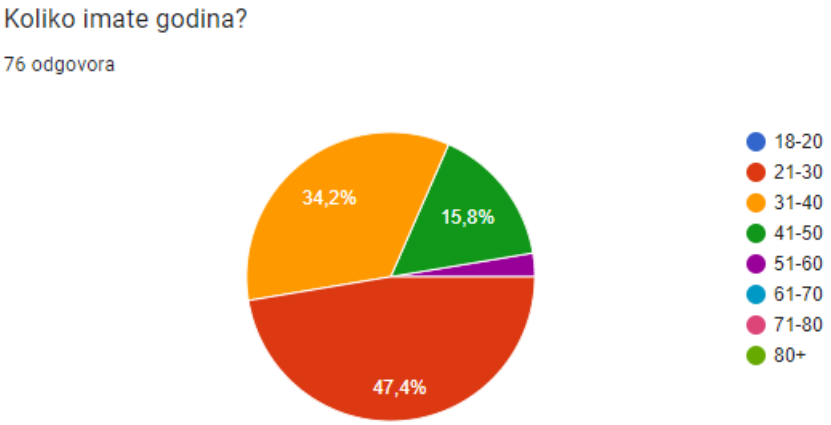
Kako bi se utvrdilo je li sustav za događaje manjeg opsega zaista potreban, provedena je anonimna anketa pomoću Google obrasci (engl. *Google Forms*) sustava.

2.4.1. Provođenje ankete

U anketi je postavljeno 8 pitanja koja ispitanike ispituju o postojećim rješenjima koja koriste za pronalazak i promoviranje manjih događaja, zadovoljstvu korištenja tih rješenja i mišljenju o predloženom rješenju. U sljedećem potpoglavlju prikazani su i objašnjeni rezultati ankete.

2.4.2. Rezultati ankete

U anketi je sudjelovalo 76 ispitanika životne dobi od 21 do 60 godina, što je vidljivo na slici (Slika 2.1).

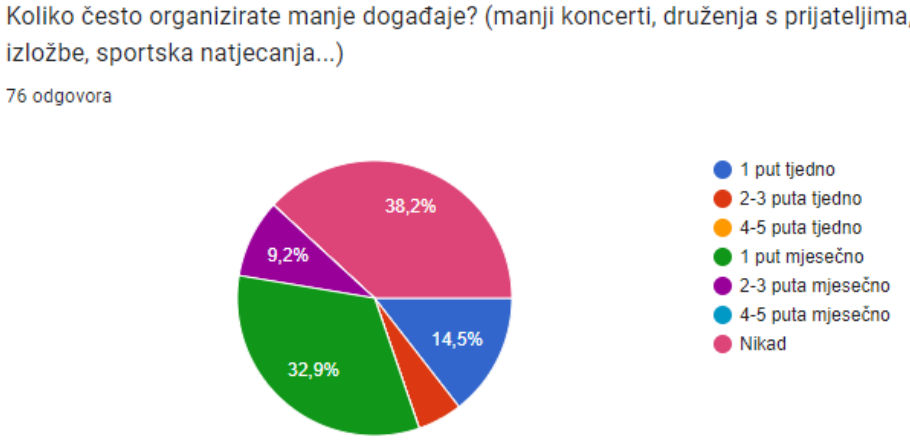


Slika 2.1 Rezultat prvog pitanja

Za uvid u navike prisustvovanja događajima, ispitanike se pita koliko često prisustvuju manjim događajima. Time se ujedno dobiva i odgovor koliko često bi netko mogao pretraživati događaje u kojima želi sudjelovati.

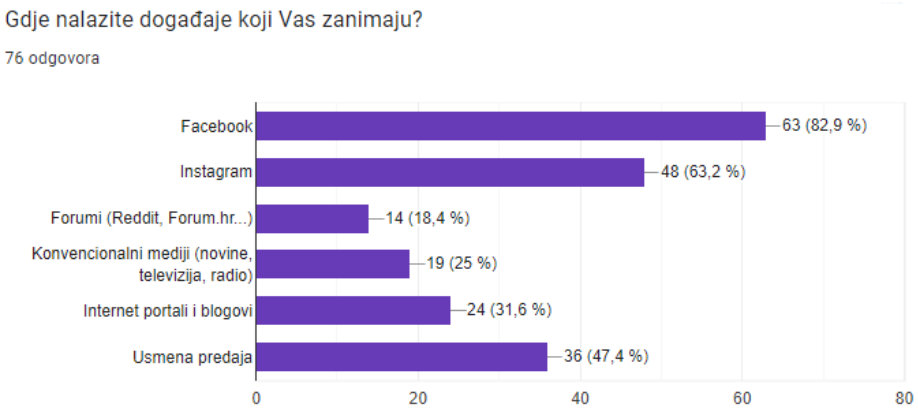
Slika 2.2 Rezultat drugog pitanja

Slično prethodnom pitanju, ispituje se i koliko često ispitanici organiziraju manje događaje.



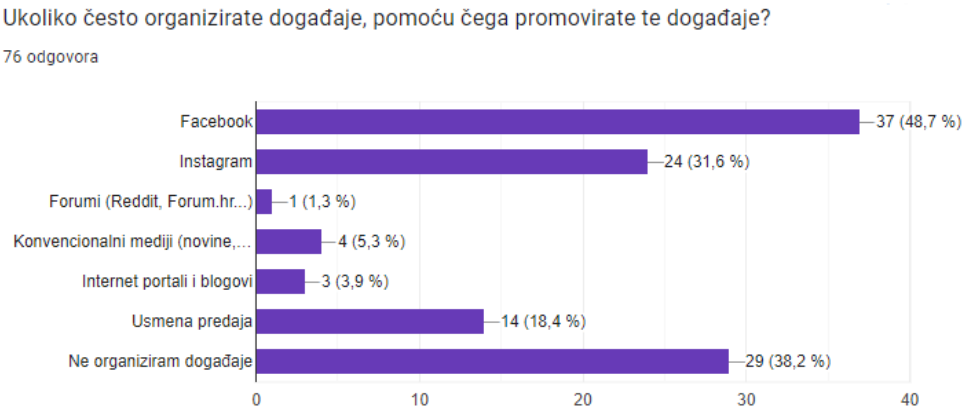
Slika 2.3 Rezultat trećeg pitanja

Od ispitanika se traži da odaberu trenutno postojeća rješenja koja koriste kako bi pronašli događaje. Na prvome mjestu stoji Facebook koji koristi 82,9% ispitanika, što je očekivan rezultat s obzirom na iznimno velik broj korisnika Facebooka u svijetu. Na slici (Slika 2.4) su prikazana postojeća rješenja koja ispitanici koriste pri traženju događaja i postoci korištenja pojedinačnog rješenja među ispitanicima.



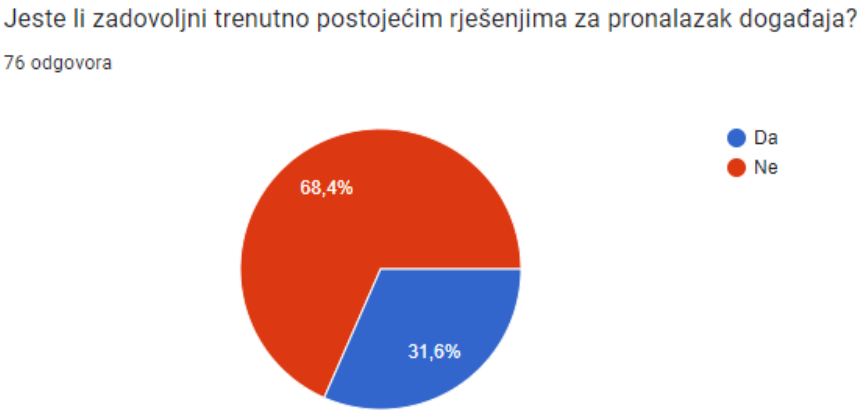
Slika 2.4 Rezultat četvrtog pitanja

Uz odabir rješenja za pronalazak događaja, ispitanici biraju i postojeća rješenja za promociju događaja koji su u njihovoj organizaciji. Kao i na prethodnom pitanju, prvo mjesto ponovno drži Facebook sa 48,7% ispitanika. Među ponuđenim odgovorima nalazi se odgovor „Ne organiziram događaje“ u slučaju da ispitanik ne organizira događaje pa ih nema potrebu niti promovirati. Iz tog razloga postoci korištenja su niži u odnosu na prethodne rezultate. Na slici (Slika 2.5) je prikazana raspoređenost postojećih rješenja za promoviranje događaja među ispitanicima.



Slika 2.5 Rezultat petog pitanja

Ispituje se i zadovoljstvo korištenja prethodno navedenih trenutno postojećih rješenja, gdje je 68,4% ispitanika izrazilo da je nezadovoljno postojećim rješenjima, što je vidljivo na slici (Slika 2.6).



Slika 2.6 Rezultat šestog pitanja

Kako bi se utvrdilo je li predloženo rješenje zaista potrebno, ispituje se misli li ispitanik da bi aplikacija s isključivim fokusom na događaje olakšala njegov pronalazak događaja koji ga zanimaju. Uz to, ispituje se i mišljenje bi li ta ista aplikacija drugima olakšala pronalazak događaja koje organizira ispitanik. U oba slučaja 66 ispitanika smatra da bi takvo rješenje olakšalo, a rezultati su vidljivi na slici (Slika 2.7).

Slika 2.7 Rezultati sedmog i osmog pitanja

Iz ankete se dolazi do zaključka da ispitanici koriste više različitih alata i izvora kako bi saznali za događaje u svojoj blizini ili kako bi promovirali događaje koje organiziraju. Većina ispitanika izrazila je nezadovoljstvo postojećim rješenjima i smatra da bi novo rješenje olakšalo njihovu pretragu, ali i pretragu drugih korisnika za događajima.

3. Arhitektura sustava

U ovom poglavlju opisana je arhitektura sustava, gdje se opisuje proces međusobne interakcije pojedinih slojeva u sustavu. Za svaki sloj navodi se njegov sastav i uloga. Uz to, opisane su sve tehnologije i programski jezici korišteni za realizaciju pojedinačnog sloja sustava.

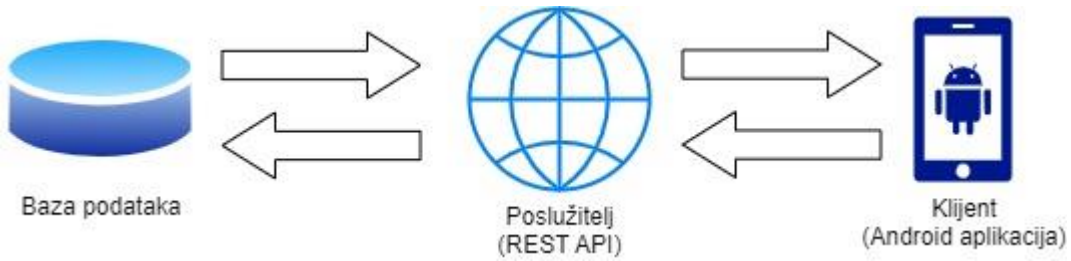
3.1. Skica i opis arhitekture sustava

Arhitektura kojom je izrađen ovaj sustav naziva se klijentsko-poslužiteljska arhitektura, a sastoji se od tri različita sloja:

- podatkovni sloj
- poslužiteljski sloj
- klijentski sloj.

Klijentsko-poslužiteljska arhitektura je model u kojem poslužitelj isporučuje i upravlja većinom resursa i usluga koje zahtjeva klijent. Komunikacija između klijenta i poslužitelja odvija se putem internetske veze, gdje klijent poslužitelju šalje zahtjeve za određenim resursima, a poslužitelj obrađuje te zahtjeve i klijentu dostavlja tražene resurse.

Podatkovni sloj sastoji se od baze podataka koju poslužitelj koristi kako bi spremao podatke koje dijeli klijentima. Poslužiteljski sloj predstavlja aplikacijsko programsko sučelje (engl. *application programming interface*, skraćeno API) koje komunicira s bazom podataka i prema potrebi upravlja njezinim podacima te ih šalje klijentima. Klijentski sloj je Android mobilna aplikacija koja dohvaća podatke s poslužiteljima i prikazuje ih korisniku na smislen i strukturiran način.



Slika 3.1 Skica arhitekture sustava

Na slici (Slika 3.1) je prikazana skica arhitekture sustava, a u nastavku su opisane tehnologije i programski jezici korišteni za implementaciju navedenih komponenti sustava.

3.2. Korištene tehnologije i jezici

U ovom poglavlju detaljno su opisane tehnologije korištene za izradu sustava. Uz to, navedeni su i različiti programski jezici pomoću kojih je razvijen sustav.

3.2.1. Podatkovni sloj

Baza podataka je organizirani skup podatkovnih zapisa pohranjenih na računalu. One omogućavaju trajno spremanje i manipuliranje podacima te su lako dostupne korisnicima i aplikacijama. Postoji više vrsta bazi podataka, a vrsta koja je korištena u ovom radu naziva se relacijska baza podataka. Relacijske baze podataka su skupovi podataka gdje su sami podaci organizirani u unaprijed određenim odnosima. Podaci su pohranjeni u jednoj ili više različitih tablica koje mogu biti međusobno povezane.

Za izradu baze podataka korišten je *Structured Query Language*, skraćeno SQL. SQL je standardizirani programski jezik koji se koristi za upravljanje relacijskim bazama podataka i izvršavanje raznih operacija nad podacima spremljenim u bazi.

Relacijska baza korištena u ovom radu je Microsoft SQL Server (skraćeno MSSQL), napravljen od strane Microsofta.

3.2.2. Poslužiteljski sloj

Za implementaciju aplikacijskoga programskog sučelja korišten je ASP.NET Core Web API. To je razvojni okvir razvijen od strane Microsofta koji služi za izgradnju HTTP usluga kojima se može pristupiti s bilo kojeg klijenta.

Komunikacija između API-ja i klijenata, koji mu žele pristupiti, odvija se putem *Hypertext Transfer Protocol* (skraćeno HTTP). HTTP je protokol koji putem mreže omogućava prijenos podataka između poslužitelja i klijenata. Komunikacija započinje u trenutku kada klijent pošalje neki zahtjev (engl. *request*) poslužitelju, a završava kada poslužitelj vrati odgovor s traženim sadržajem (engl. *response*). Zahtjevi koje klijent šalje poslužitelju su u obliku usklađenog lokatora sadržaja (engl. *Uniform Resource Locator*, skraćeno URL). URL

sadrži tri ključne informacije, a to su HTTP metoda, jedinstveni identifikator resursa (engl. *Uniform Resource Identifier*, skraćeno URI) i verziju protokola [2].

HTTP metode su naredbe koje javljaju poslužitelju koju operaciju moraju izvršiti nad resursom. Postoji više različitih HTTP metoda, a neke od najkorištenijih su:

- GET (dohvaćanje resursa)
- POST (izrada novog resursa)
- PUT (ažuriranje postojećeg resursa)
- DELETE (brisanje postojećeg resursa).

HTTP protokol prilikom vraćanja odgovora klijentu šalje i odgovarajuće statusne kodove koji naznačuju je li određeni HTTP zahtjev uspješno izvršen. Kodovi su definirani prema RFC 9110 standardu [2], prikazuju se numeričkim vrijednostima, a dijele se na 5 vrsta:

- informativni odgovori (vrijednosti od 100 do 199)
- uspješni odgovori (vrijednosti od 200 do 299)
- poruke za preusmjeravanje (vrijednosti od 300 do 399)
- pogreške na klijentu (vrijednosti od 400 do 499)
- pogreške na poslužitelju (vrijednosti od 500 do 599).

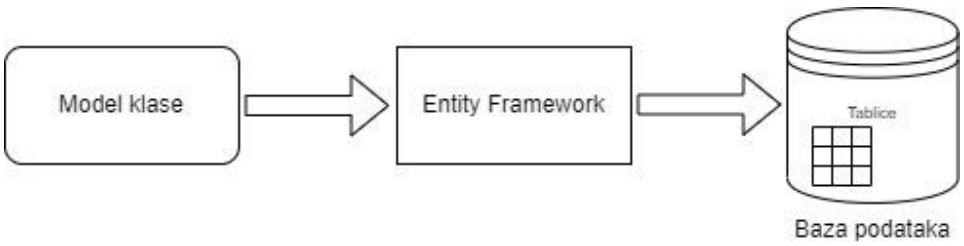
Sam API je rađen prema *representational state transfer* (skraćeno REST) arhitekturi [3]. REST je vrsta arhitekture API-ja, a da bi se neki API smatrao RESTful, mora zadovoljavati sljedeće kriterije:

- Klijentsko-poslužiteljska arhitektura koja se sastoji od klijenta, poslužitelja i resursa (podaci spremljeni u bazi podataka).
- Ujednačeno sučelje kroz korištenje HTTP metoda putem URL-a za manipulaciju resursima.
- Mogućnost pamćenja podataka koji pojednostavljuju interakcije klijent-poslužitelj i čine komunikaciju bržom i efikasnijom. Pogodno kod uzastopnog traženja istog resursa od strane klijenta.
- Bez stanja (engl. *stateless*), što znači da se ne pamte nikakve informacije o klijentima. Svi zahtjevi su odvojeni i nepovezani.

Za povezivanje baze podataka i API-ja korišten je *Entity Framework*, objektno relacijska tehnologija (engl. *Object Relational Mapping*, skraćeno ORM) za mapiranje objekata iz programskog koda s tablicama u bazi podataka, razvijen od strane Microsofta. *Entity Framework* je koristan alat prilikom izrade API-ja zato što se pomoću njega baza i API mogu jednostavno spojiti i komunicirati, bez potrebe za pisanjem ikakvih SQL naredbi ili procedura. Postoje tri pristupa za razvoj aplikacije pomoću Entity Frameworka, a to su:

- *Database-first*
- *Code-First*
- *Model-First.*

U ovom je radu korišten *Code-first* pristup u kojem se kreće od kodiranja entiteta (model klase) i *context* klasa. Pomoću tih klasa generiraju se SQL skripte koje sadrže sve potrebne naredbe za izradu baze podataka i tablica koje odgovaraju prethodno napisanom modelu. Na posljetku, skripta se izvršava na bazi korištenjem migracijskih naredbi [4]. Cijeli proces je skiciran i vidljiv na slici (Slika 3.2)



Slika 3.2 Skica *Code-first* pristupa pri korištenju Entity Frameworka

REST API može u više različitih oblika vraćati resurse na zahtjev, a onaj najčešći i korišten u ovom projekt je *JavaScript Object Notation* (skraćeno JSON). JSON je standardizirani format za razmjenu podataka koji koristi ključ-vrijednost parove i nizove za spremanje i prikaz podataka. U nastavku je prikazan primjer objekta student koji sadrži atribute ime i prezime, prikazan u JSON formatu.

```
{
"studenti":[
    {"ime":"Pero", "prezime":"Perić" },
    {"ime":"Marko", "prezime":"Marić"}
]
}
```

3.2.3. Klijentski sloj

Za implementaciju klijentskog sloja sustava razvijena je mobilna aplikacija za Android operacijski sustav pomoću Kotlin programskog jezika. Aplikacije za Android je moguće razvijati i pomoću Java programskog jezika, ali zbog određenih prednosti koje su opisane u nastavku, za ovaj je rad izabran Kotlin. Dakle, Kotlin je *multi-platform*, statički tipiziran programski jezik proizveden od strane JetBrainsa i Googlea. Dizajniran je tako da nudi potpunu interoperabilnost s Javom, a može se koristiti za brojne vrste razvoja, kao što su serveri, web-servisi i Android aplikacije [5]. Također, koristi koncepte objektno orijentiranog i funkcijskog programiranja, a sintaksa je čišća, jednostavnija i kraća, sličnija govornom jeziku.

Za razvoj Android mobilnih aplikacija postoji velik broj tehnologija i biblioteka koje programerima olakšavaju i ubrzavaju razvoj te omogućavaju implementaciju raznih funkcionalnosti. Takve su tehnologije korištene i u ovome radu:

- Retrofit
- Google Maps API
- Firebase Authentication
- Firebase Realtime Database
- Material Design 3.

Retrofit biblioteka je REST klijent razvijen za Javu i Kotlin programske jezike, a služi za olakšanu autentifikaciju i interakciju s API-jima. Retrofit upravlja procesima stvaranja, slanja i dohvaćanja HTTP zahtjeva i odgovora. Prednosti kod korištenja Retrofita su brzina, jednostavnost implementacije i razumijevanja te podrška za sinkrone i asinkrone mrežne pozive.

Google Maps API je API koji nudi Android za implementaciju Google karti u mobilnu aplikaciju. Pomoću njega prikazuje se karta unutar aplikacije, a također može pružati i dodatne informacije za lokacije na karti te podržava interakciju korisnika.

Firebase Authentication je *Software Development Kit* (skraćeno SDK), odnosno kolekcija razvojnih alata koja služi za verifikaciju korisnika unutar aplikacije [6]. Verifikacija se može odvijati putem više različitih načina, kao što su verifikacija putem elektroničke pošte i lozinke, broja mobilnog telefona i putem korisničkih računa popularnih društvenih mreža

Facebooka i Twittera. Za ovaj rad je izabrana verifikacija putem elektroničke pošte i lozinke. Prednosti kod implementacije verifikacije putem Firebasea su brzina i jednostavnost ugradnje i korištenja te velik broj načina na koji se korisnik može verificirati. Još jedna velika prednost je što nije potrebno razvijati poslužitelja za verifikaciju, već sam Firebase generira bazu podataka koju potom koristi za spremanje i čitanje korisničkih podataka.

Firebase Realtime Database je *NoSQL* baza podataka u oblaku (engl. *cloud*), gdje u usporedbi s relacijskom bazom podataka ima različite optimizacije i funkcionalnosti. Podaci u toj bazi su spremljeni u JSON formatu te se sinkroniziraju na svim klijentima u stvarnom vremenu. Firebase Realtime Database ima široke primjene, a u ovom radu koristi se za implementaciju funkcionalnosti za razgovor između verificiranih korisnika.

Material Design 3 je prilagodljiv sustav smjernica, komponenti i alata koji podržavaju najbolju praksu dizajna korisničkog sučelja u Android i iOS mobilnim te web-aplikacijama, razvijen od strane Googlea. [7]

4. Implementacija sustava

Ovo poglavlje bavi se opisivanjem koraka u implementaciji osmišljenog sustava. Za svaki sloj sustava posebno se prikazuje način na koji je implementiran i koje su prethodno predstavljene tehnologije korištene pri izradi funkcionalnosti.

4.1. Podatkovni sloj

S obzirom na to da se podatkovni sloj sastoji isključivo od baze podataka, implementacija ovog sloja vrlo je jednostavna, a uključuje izradu baze i objavu te iste baze podataka na Microsoft Azure servisu.

Azure je javna *cloud* platforma koja nudi velik broj različitih servisa koji se mogu koristiti za analizu, virtualno računalstvo, prostor i umrežavanje. U ovom radu koristi se za objavu baze podataka zato da joj se može pristupiti pomoću poslužiteljskog sloja. Azure se također koristi i za objavu poslužiteljskog sloja, što će se detaljnije prikazati u nastavku.

4.2. Poslužiteljski sloj

Ovim poglavljem predstavlja se implementacija poslužiteljskog sloja sustava. S implementacijom se definira cijeli proces dizajniranja poslužitelja uz opisivanje i objašnjavanje svih njegovih komponenti.

4.2.1. Pristupne točke

HTTP pristupne točke u poslužiteljskom sloju su ciljane URL adrese na nekoj web-aplikaciji ili servisu koji se koriste za slanje zahtjeva od strane klijenta prema nekom servisu. Kod javnih API-ja postoje dokumentacije koje je potrebno pročitati kako bi se znalo koje pristupne točke taj servis sadrži i na koji se način koriste. Svaka pristupna točka sastoji se od nekoliko dijelova, a to su:

- Vrsta HTTP protokola koja se koristi – opcije su http ili https, a razlika između njih je ta da https koristi enkripciju i verifikaciju kako bi osigurao obične HTTP pozive
- HTTP metoda
- Mrežna lokacija web-servera.

Neke pristupne točke mogu sadržavati URI koji služi kao putanja do određenog resursa pa se u tom slučaju, u URL-u zahtjeva nalazi i taj jedinstveni identifikator. Ako pristupna točka ne zahtijeva URI, web-servis razlikuje zahtjeve prema HTTP metodi koju je klijent poslao prilikom zahtjeva. Kroz pristupne točke je moguće slati i razne parametre ako je to na njoj definirano, a ti parametri također se nalaze u URL-u.

Ispod je prikazan primjer jedne pristupne točke, korištene od strane klijenta za dohvaćanje popisa nadolazećih događaja.

`https://zavrsniradapi.azurewebsites.net/api/Event/GetUpcomingEvents`

U nastavku je opisan i objašnjen proces dizajna i implementacije API-ja, čiji je rezultat javno dostupan API servis s pristupnim točkama koje će se koristiti na klijentu.

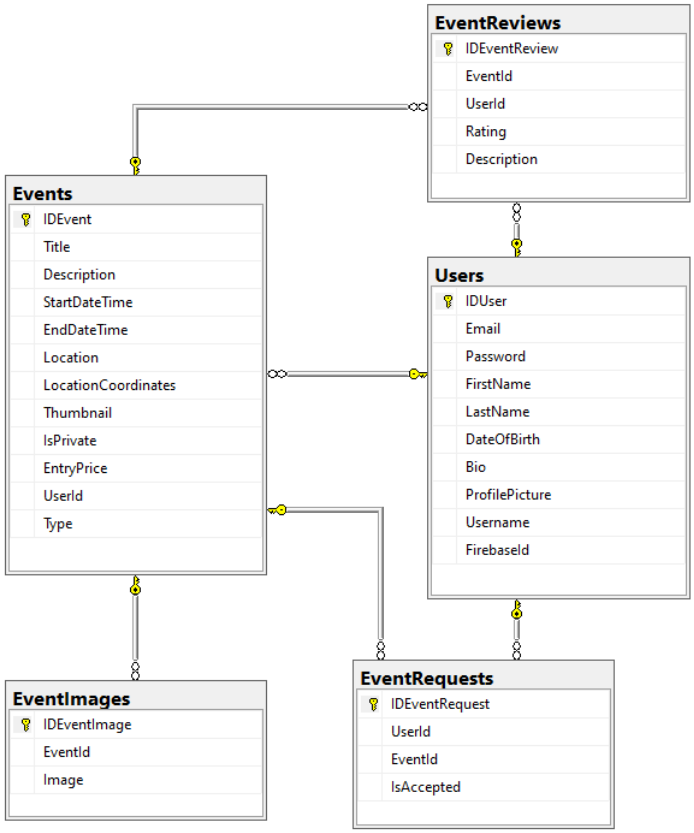
4.2.2. Dizajn API-ja

Za implementaciju i dizajn poslužiteljskog sloja korišten je Microsoft Visual Studio 2022 razvojno okruženje, a prvi je korak izrada projekta prema ASP.NET Core Web API šabloni. Nakon uspješno izrađenog projekta, potrebno je spojiti aplikaciju s prethodno izrađenom bazom podataka putem njezinog *connection stringa*. To je referenca koja sadrži informacije kao što su putanja do servera, na kojem se nalazi baza, naziv baze i korisničke informacije. *Connection string* se unutar projekta zapisuje u `appsettings.json` datoteku na način ključ-vrijednost, gdje je ključ unikatan naziv pomoću kojeg će se u nastavku dohvaćati ta referenca, a vrijednost je *connection string*.

Nakon uspješno postavljene reference na bazu podataka, kreće konfiguracija Entity Framework ORM tehnologije korištenjem *Code-first* pristupa. Prvi korak kod ovog pristupa je kreiranje model klasi, odnosno izrada objekta koji će ujedno biti i tablice u bazi podataka. U nastavku je navedena i opisana svaka klasa modela korištena za ovaj projekt, a na slici (Slika 4.1) se nalazi dijagram koji prikazuje sve navedene objekte i veze između njih.

- `AppUser` – sastoji se od podataka koji služe za verifikaciju nekog korisnika i podataka koji prikazuju detalje tog korisnika.

- `Event` – sadrži podatke o svim potrebnim detaljima nekog događaja, kao što su naziv, vrijeme i mjesto događaja, vrsta, privatnost i cijena ulaznice. Također, sadrži i referencu na `AppUser` klasu, gdje se referencira korisnik koji organizira taj događaj.
- `EventImage` – sadrži sliku za određeni događaj i referencu na taj događaj. Iz razloga što jedan događaj može sadržavati više slika, napravljena je posebna tablica, odnosno objekt.
- `EventRequest` – sastoji se od reference na korisnika koji je zatražio zahtjev, reference na događaj za koji je zahtjev zatražen te atributa koji označava je li zahtjev prihvaćen.
- `EventReview` – sadrži podatke o recenzijama za neki događaj, a sastoji se od reference na događaj koji se ocjenjuje, ocjene, reference na korisnika koji radi recenziju i opisa recenzije.



Slika 4.1 Dijagram odnosa entiteta

Nakon izrađenih klasa modela, potrebno je napraviti *context* klasu. To je najvažnija klasa prilikom rada s Entity Frameworkom iz razloga što predstavlja sjednicu (engl. *session*) s

povezanom bazom podataka [4]. Putem nje moguće obavljati CRUD operacije, odnosno operacije stvaranja, čitanja, ažuriranja i brisanja nad prethodno kreiranim objektima. Sastoji se od atributa, gdje je svaki atribut kolekcija od jedne klase modela.

```
public class AppDbContext : DbContext
{
public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options)
{
}

public DbSet<AppUser> Users { get; set; }
public DbSet<SocialMediaLink> SocialMediaLinks { get; set; }
public DbSet<Event> Events { get; set; }
public DbSet<EventImage> EventImages { get; set; }
public DbSet<EventStory> EventStories { get; set; }
public DbSet<EventReview> EventReviews { get; set; }
public DbSet<EventRequest> EventRequests { get; set; }
}
```

Kod 4.1 AppDbContext klasa

Za konačno kreiranje tablica u bazi podataka, koje odgovaraju prethodno izrađenim klasama, potrebno je napisati dvije naredbe u Package Manager konzolu. Prva je naredba `Add-Migration` kojom se generiraju SQL naredbe za izradu svih potrebnih tablica i relacija prema klasama modela. Druga naredba je `Update-Database` koja izvršava prethodno kreiranu migraciju i primjenjuje promjene na shemu baze podataka [4].

S uspješno generiranim tablicama u bazi podataka i *context* klasom, pomoću koje je moguće pristupiti i manipulirati podacima u bazi, kreće izrada API-ja.

REST API razvijen je korištenjem obrasca repozitorija (engl. *repository pattern*) koji je jedan od oblikovnih obrazaca (engl. *design patterns*) u razvoju programskih rješenja. Oblikovni obrasci su opća, ponovljiva rješenja problema koji se često pojavljuju u dizajnu softvera. Prednost kod korištenja oblikovnih obrazaca je ubrzavanje razvojnog procesa uz pomoć ispitanih, dokazanih razvojnih paradigmi. U ovom radu koristi se obrazac repozitorija kako bi se razdvojili slojevi pristupa podacima i poslovne logike aplikacije. Time se pomoću jednostavnih metoda može pristupiti podacima iz baze podataka unutar poslovne logike aplikacije. Prednost kod obrasca repozitorija je što pruža bolju održivost koda u budućnosti.

Obrazac repozitorija sastoji se od klasa repozitorija koje sadrže poslovnu logiku dohvaćanja podataka iz podatkovnog sloja te sučelja koja definiraju ponašanje tih klasa.

Za početak, potrebno je napisati sučelja koja određuju metode koje će klase repozitorija implementirati. Za svaku klasu modela kreirano je sučelje i klasa repozitorija. U kodu (Kod 4.2) prikazano je sučelje `IEventRepository` koje definira metode za dohvaćanje, kreiranje, ažuriranje i brisanje entiteta klase `Event` iz baze podataka.

```
public interface IEventRepository
    {
        Task<IEnumerable<Event>> GetAllEvents();
        Task<IEnumerable<Event>> GetUpcomingEvents();
        Task<IEnumerable<Event>> GetEventsForLocation(string
            location);
        Task<IEnumerable<Event>>
            GetUpcomingEventsForType(string type);
        Task<IEnumerable<Event>> GetCurrentEventsForUser(int
        id);
        Task<IEnumerable<Event>> GetPastEventsForUser(int
        id);
        Task<Event> GetEvent(int id);
        Task<Event> AddEvent(Event newEvent);
        Task<Event> UpdateEvent(Event newEvent);
        Task DeleteEvent(int id);

    }
```

Kod 4.2 IEventRepository sučelje

Metode u sučelju, kao tip podatka koji vraćaju, sadrže klasu `Task`. To je klasa u C# programskom jeziku koja označava operaciju koja može i ne mora vraćati vrijednost, a obično se izvršava asinkrono. Ako metoda vraća neku vrijednost, uz ključnu riječ `Task` unutar izlomljenih zagrada potrebno je definirati tip podatka koji ta metoda vraća. Sve metode na poslužitelju izvršavat će se asinkrono kako bi poslužitelj mogao paralelno obrađivati više zahtjeva, što dovodi do poboljšanog vremena odziva i boljeg korištenja resursa sustava.

Nakon definiranja sučelja, sljedeći je korak izrada klase repozitorija za svako od prethodno kreiranih sučelja. Svaka klasa implementira jedno sučelje i sve njegove metode, a da bi neka klasa implementirala neko sučelje, potrebno je uz ime klase staviti dvotočku i naziv sučelja

koje se implementira. Klase imaju mogućnost i implementacije više različitih sučelja, a u tom slučaju imena sučelja je potrebno odvojiti zarezom.

Unutar klase repozitorija potrebno je i kreirati instancu objekta `AppDbContext` klase kako bi se moglo pristupiti podacima iz baze. Način na koji se kreira instanca tog objekta je putem konstruktora klase repozitorija, tj. metode zadužene za kreiranje objekta definiranog pomoću neke klase. Zatim se kreće na implementaciju pojedinačnih metoda sučelja, u kodu (Kod 4.3) je prikazana metoda AddEvent koja služi za kreiranje novog događaja u bazi podataka.

```
public async Task<Event> AddEvent(Event newEvent)
        {
            var result = await
appDbContext.Events.AddAsync(newEvent);
            await appDbContext.SaveChangesAsync();
            return result.Entity;
        }
```

Kod 4.3 AddEvent metoda za kreiranje novog događaja

Modifikator pristupa za ovu metodu, a i sve ostale metode u klasi repozitorija, je `public` što označava da se s bilo kojeg mjesta u programskom kodu može pristupiti toj metodi. Nadalje, metoda je označena i ključnom riječi `async` , što znači da se izvršava asinkrono. Tip podataka koji se vraća je `Task<Event>` zbog toga što se nakon kreiranja novog događaja, taj novokreirani događaj vraća kao odgovor korisniku kako bi se potvrdilo da je zahtjev uspješno izvršen. Kao parametar metodi šalje se objekt događaja koji korisnik želi dodati u bazu. U tijelu metode može se primijetiti korištenje prethodno navedenog `appDbContext` objekta i njenog atributa liste svih događaja iz baze podataka po imenu `Events`. Metoda radi na način da se toj listi dodaje novi događaj pomoću ugrađene metode `AddAsync` koja asinkrono dodaje događaj u popis događaja dohvaćenih iz baze podataka. Nakon dodavanja, potrebno je spremiti novu promjenu u bazi podataka pomoću ugrađene asinkrone metode `SaveChangesAsync` na `appDbContext` objektu. Uz pozivanje asinkronih metoda, potrebno je dodati operator `await` koji obustavlja procjenu priložene asinkrone metode, dok se asinkrona operacija predstavljena njegovim operandom ne završi. Kada se `await` operator doda metodi koja predstavlja već gotovu operaciju, pri završetku se vraća rezultat, ako postoji, bez obustave metode u kojoj se operacija izvršava [8]. Na kraju, kao što je prethodno navedeno, metoda vraća rezultat novokreiranog entiteta objekta `Event`.

Nakon implementacije svih sučelja, sljedeći korak za realizaciju REST API servisa je izrada kontroler klasi (engl. *controller*). Kontroleri su klase u web API servisima koje obuhvaćaju metode koje će se kasnije pozivati s klijenta u obliku zahtjeva. U sebi sadrže poslovnu logiku za obradu zahtjeva klijenta i služe kao poveznica između klijenta i baze podataka putem prethodno kreiranih repozitorija. Kako bi neka klasa bila *controller* klasa, mora naslijediti klasu `ControllerBase`. Uz to, potrebno je postaviti atribut `[ApiController]` iznad samog naziva klase. Još jedan nužan atribut za *controller* klasu je `[Route("api/[controller]")]`. Pomoću njega moguće je prilagoditi rutu po kojoj se pristupa toj klasi putem zahtjeva klijenta, a unutar zagrada upisan je oblik rute [9]. U ovom slučaju, oblik rute je vrlo jednostavan, a sastoji se od ključne riječi „api" i naziva kontrolera kojem se želi pristupiti. U ovom radu rute su postavljene na jednak način na svakoj od *controller* klasa.

U kodu (Kod 4.4) prikazana je metoda `GetEvents` koja na zahtjev klijenta poziva metodu iz repozitorija za dohvaćanje događaja te klijentu vraća odgovor sa zahtijevanim resursom.

```
[HttpGet]
[Route("All")]
 public async Task<ActionResult> GetEvents()
    {
         try
         {
             return Ok(await eventRepo.GetAllEvents());
         }
         catch (System.Exception)
         {
             return
StatusCode(StatusCodes.Status500InternalServerError,
            "Error retrieving data from the database");
            }
        }
```

Kod 4.4 GetEvents metoda za dohvaćanje događaja

Postoje dva bitna atributa koja svaka metoda u kontroleru mora sadržavati, a to su HTTP metoda, kojom se poziva, i ruta preko koje se poziva. Oba atributa definiraju se u uglatim zagradama iznad naziva metode. U kodu (Kod 4.4) definirana je HTTP metoda GET u obliku `[HttpGet]` , što znači da joj se može pristupiti isključivo slanjem GET metode u zahtjevu klijenta [9]. Svakoj metodi moguće je odrediti i proizvoljnu rutu, što se koristi u slučaju ako

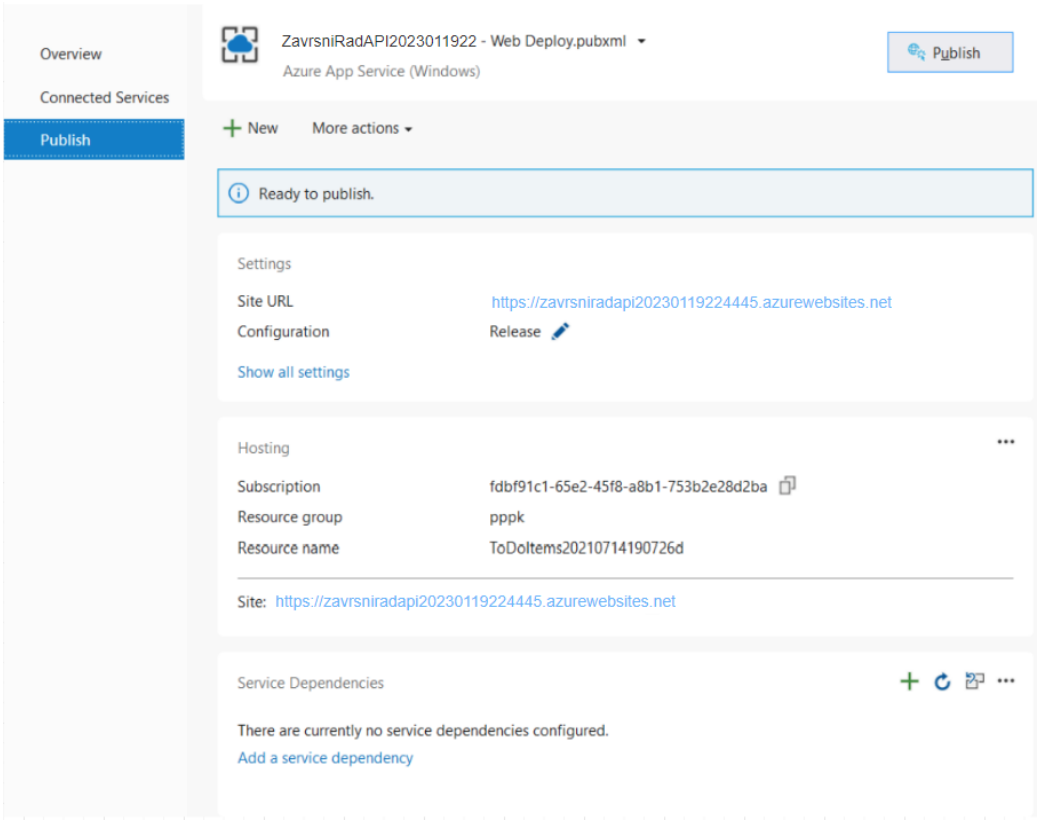
unutar jednog kontrolera postoji više metoda s istim parametrima ili istom definiranom HTTP metodom. S obzirom na to da prilikom slanja zahtjeva klijent u URL-u može dodati i neke parametre koje šalje zajedno sa zahtjevom, u atributu rute moguće je navesti parametre koje klijent treba poslati, a koji se prosljeđuju metodi. Primjer metode koja na navedeni način prima parametre nalazi se u kodu (Kod 4.5). Nazvana je `GetEventsForType`, služi za dohvaćanje događaja za određeni tip, a parametar koji prima je tip događaja prikazan u teksutalnom obliku.

```
[HttpGet]
        [Route("ForType/{type}")]
      public async Task<ActionResult>
      GetEventsForType(string type)
        {
            try
            {
                return Ok(await
eventRepo.GetUpcomingEventsForType(type));
            }
            catch (System.Exception)
            {
                return
StatusCode(StatusCodes.Status500InternalServerError,
                    "Error retrieving data from the
database");
            }
        }
```

Kod 4.5 Metoda GetEventsForType

Kako bi se klijentu pri vraćanju odgovora vratio rezultat s pripadajućim HTTP statusnim kodom, korištena je metoda `return Ok()` koja kao parametar prima kod koji se treba izvršiti, a vraća rezultat koda koji je poslan kao parametar te statusni kod 200 koji označava da je zahtjev uspješno izvršen. Također, korištena je i metoda `return StatusCode()` koja putem statusnog koda i poruke, koji su postavljeni kao parametri metode, obavještava klijenta da zahtjev nije uspio. Tim načinom implementirane su sve metode u svim kontrolerima unutar ovog REST API-ja, a prednost ovog pristupa je da klijent dobiva informaciju o uspješnosti svoga zahtjeva i opis potencijalne pogreške koja je nastala prilikom izvršavanja tog zahtjeva.

Nakon implementacije kontrolera, potrebno je objaviti ovaj API servis kako bi mu klijenti mogli pristupiti putem interneta i koristiti njegove funkcionalnosti. Postoji više različitih načina za objavu servisa, a u ovom radu korišten je Azure Web Service, platforma za objavu web-servisa razvijena od strane Microsofta. Objava web-servisa se plaća stoga je potrebno imati Microsoft Azure korisnički račun koji na sebi ima aktiviranu pretplatu ili postavljenu kreditnu karticu. Objava nekog servisa je vrlo jednostavna, a obavlja se direktno iz Microsoft Visual Studio okruženja u kojem je servis razvijen. Desnim klikom miša na naziv projekta otvara se alatni prozor na kojem je potrebno kliknuti na gumb Objavi (engl. *Publish*). Zatim se otvara prozor gdje se unose korisnički podaci Azure računa pa se praćenjem koraka kreirani servis objavljuje na internet. Kao rezultat uspješne objave, dobiva se web-adresa putem koje je moguće tom API-ju pristupiti, a koja će se koristiti od strane klijenta za slanje zahtjeva. Na slici (Slika 4.2) nalazi se prozor za objavu servisa u Microsoft Visual Studio okruženju.



Slika 4.3 Prozor za objavu web-servisa

4.3. Klijentski sloj

Implementacija klijentskog sloja ostvarena je u Android Studio razvojnom okruženju pomoću Kotlin programskog jezika. Zbog velikoga broja različitih verzija Android operacijskog sustava, prvi korak pri izradi projekta je odabir najstarije podržane verzije. Svaka nova verzija Android sustava sa sobom nosi nove funkcionalnosti u kontekstu hardvera i softvera, ali ipak ne podržavaju svi Android uređaji najnovija ažuriranja operacijskog sustava. Zbog toga se razvojni inženjer pri izradi mobilne aplikacije za Android mora odlučiti koju će najstariju verziju operacijskog sustava ta aplikacija podržavati. Taj odabir rezultira postotkom korisnika Android uređaja koji će uspješno moći instalirati i koristiti tu aplikaciju. Za ovaj rad, za najstariju podržanu verziju odabran je Android 6.0 po imenu Marshmallow iz razloga što će 96,2% Android korisnika biti u mogućnosti koristiti ovu aplikaciju na svojim uređajima. Na slici (Slika 4.3) prikazane su sve dosadašnje verzije Android operacijskog sustava s pripadajućim postocima distribucije na uređajima.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.4 KitKat	19	
5.0 Lollipop	21	99,3%
5.1 Lollipop	22	99,0%
6.0 Marshmallow	23	97,2%
7.0 Nougat	24	94,4%
7.1 Nougat	25	92,5%
8.0 Oreo	26	90,7%
8.1 Oreo	27	88,1%
9.0 Pie	28	81,2%
10. Q	29	68,0%
11. R	30	48,5%
12. S	31	24,1%
13. T	33	5,2%

Slika 4.4 Verzije Android operacijskog sustava s postocima distribucije

Nakon odabira Android verzije i kreiranja projekta, projektu je potrebno postaviti zavisnosti (engl. *dependency*), odnosno odrediti s kojim vanjskim tehnologijama i bibliotekama aplikacija radi. Taj dio koda piše se u `build.gradle` datoteci koja upravo služi za definiranje zavisnosti koje su onda dostupne u cijelom projektu. U kodu (Kod 4.6) je prikazan način dodavanja svih potrebnih biblioteka za rad s Firebase tehnologijom.

```
implementation 'com.google.firebase:firebase-core:21.1.1'
    implementation 'com.google.firebase:firebase-auth:21.1.0'
    implementation 'com.google.firebase:firebase-storage-
ktx:20.1.0'
    implementation 'com.google.firebase:firebase-database-
ktx:20.1.0'
    implementation 'com.google.firebase:firebase-
messaging:23.1.1'
```

Kod 4.6 Kod za dodavanje Firebase biblioteka

4.3.1. Korisničko sučelje

U ovom dijelu, opisane su sve korištene komponente za izradu korisničkog sučelja aplikacije te njihova namjena u sučelju. Glavna komponenta pri izradi Android aplikacija je aktivnost (engl. *activity*) koja predstavlja prozor koji sadrži komponente korisničkog sučelja, a sastoji se od tlocrta (engl. *layout*) u koji se postavljaju komponente te `Activity` klase koja sadrži poslovnu logiku za prikazivanje podataka na prozoru.

Za izradu tlocrta neke aktivnosti moguće je koristiti tri pristupa, a to su izrada tlocrta pomoću „*drag and drop*“ dizajnera ugrađenog u Android Studio razvojno okruženje, putem pisanja posebne vrste XML koda namijenjene za Android ili putem pisanja koda u `Activity` klase. Za ovaj rad odabran je način pisanja XML koda kako bi se kreirali trocrti, a za povezivanje komponenti iz tlocrta s programskim kodom koristi se postavka `viewBinding` koja se postavlja u datoteci `build.gradle` na sljedeći način:

```
buildFeatures {
        viewBinding true
    }
```

Klase aktivnosti, koje se nalaze u ovom projektu, a koje služe da bi se podaci s poslužitelja na smislen i intuitivan način prikazale korisniku su sljedeće:

- `SignInActivity` – prva aktivnost koja se otvara prilikom pokretanja aplikacije. Koristi se za prijavu korisnika s postojećim korisničkim računom.
- `RegisterActivity` – služi za izradu novog korisničkog računa.

- `MapsActivity` – zamišljena kao glavna aktivnost aplikacije gdje se nalazi Google karta putem koje korisnik može pregledavati događaje. Također, na sebi ima navigacijske gumbe kojima se pristupa ostalim aktivnostima aplikacije.
- `EventActivity` – aktivnost koja korisniku prikazuje detalje nekog događaja. Ako je događaj završio, prikazuje i njegove recenzije.
- `CreateActivity` – koristi se za izradu novih događaja.
- `EditEventActivity` – koristi se za uređivanje postojećih događaja.
- `AccountActivity` – aktivnost za prikazivanje detalja računa korisnika i svih njegovih trenutnih ili prošlih događaja.
- `MyChatsActivity` – popis svih razgovora.
- `ChatActivity` – aktivnost koja sadrži razgovor s nekim korisnikom.
- `EventRequestActivity` – popis zahtjeva za neki događaj. Prikazuje samo u slučaju kada je neki događaj privatan.

`SignInActivity` je vrlo jednostavna aktivnost koja se sastoji od dvije `EditText` komponente, zadužene za upisivanje teksta i gumba za prijavu. Također, sadrži i gumb za registraciju koji korisnika vodi na `RegisterActivity`.

`RegisterActivity` je formular načinjen od `EditText` komponenti koje korisniku omogućavaju upis detalja novoga korisničkog računa.

`MapsActivity` sadrži Google kartu koja popunjava cijeli ekran. Na karti su pomoću markera prikazani događaji, a klikom na marker otvara se mali prozor s kratkim informacijama o tom događaju. Također, sadrži i gumbe za navigaciju po aplikaciji. Navigacijski gumbi su prikazani `FloatingActionButton` komponentom koja je slična običnim `Button` komponentama. Svaki gumb sadrži ikonu koja označava korisniku do koje funkcionalnosti vodi.

Korisničko sučelje aktivnosti `EventActivity` je vrlo složeno zbog količine i različitih vrsti podataka koje mora prikazivati. Za prikaz tekstualnih informacija, kao što su naziv događaja, vrijeme i mjesto, tip i opis, korištena je `TextView` komponenta kojom je moguće prikazivati tekst i oblikovati taj prikaz na razne načine. Za prikaz slike događaja i svih ikona, koje se nalaze na prozoru, zadužen je `ImageView`. Recenzije događaja prikazane su uz pomoć `RecyclerView` komponente koja služi za efikasno prikazivanje velikih skupova

podataka. Zbog velikog broja komponenti, na ovom dijelu korisničkog sučelja korišten je i `ScrollView` koji korisniku omogućava vertikalno listanje (engl. *scroll*) na toj aktivnosti.

`CreateActivity` sadrži formular koji se sastoji od većeg broja `EditText` polja za upisivanje detalja novog događaja. S obzirom na to da događaji sadrže informacije datuma i vremena, za unos tog tipa informacija korištene su `MaterialDatePicker` i `MaterialTimePicker` komponente iz Material 3 biblioteke. One na intuitivan način daju korisniku mogućnost odabira datuma putem kalendara i vremena putem sata, bez potrebe za upisivanjem. Za odabir lokacije događaja koristi se Google karta. Ova aktivnost i `EditEventActivity` su identične, s jedinom razlikom da su u `EditEventActivity` sva polja popunjena informacijama događaja koji korisnik želi urediti.

`AccountActivity` za prikaz tekstualnih informacija korisničkog računa koriste se `TextView` elementi. Svaki korisnički račun sadrži i profilnu sliku koja je prikazana u `ImageView`. Uz detalje računa, prikazani su trenutni i prošli događaji tog korisnika pomoću dva `RecyclerView` elementa, gdje se svaki nalazi u svom fragmentu. Fragment predstavlja dio korisničkog sučelja koji sadrži vlastitu poslovnu logiku za prikaz podataka, a njegove prednosti su modularnost i mogućnost višekratne uporabe. Da bi fragmenti funkcionirali, moraju biti dio neke aktivnosti ili drugog fragmenta. Za navigaciju između popisa trenutnih i prošlih događaja, odnosno fragmenata u kojima se oni nalaze, koristi se `TabLayout`.

`MyChatsActivity` sadrži popis svih razgovora korisnika prikazanog pomoću RecyclerView komponente. Svaki je razgovor prikazan korisničkim imenom i slikom profila korisnika s kojim se sudjeluje u razgovoru. Na isti način dizajnirano je i korisničko sučelje `EventRequestsActivity` , gdje su umjesto razgovora prikazani zahtjevi za pristup događaju. Zahtjev se sastoji od korisničkog imena korisnika, koji je poslao taj zahtjev, gumba za prihvaćanje i gumba za odbijanje zahtjeva.

`ChatActivity` za prikaz poruka u nekom razgovoru također koristi RecyclerView, a za upis nove poruke koristi se EditText polje. Kako bi se poslane i primljene poruke mogle lako razlikovati, poruke poslane od strane korisnika poravnate su na desnoj strani ekrana, dok su primljene poruke poravnate na lijevoj strani ekrana. Uz to, pozadinska boja poslanih i primljenih poruka je drugačija.

4.3.2. Opis funkcionalnosti aplikacije

Ovaj dio opisuje način implementacije funkcionalnosti aplikacije te njihovu namjenu i zamišljeni način korištenja. Iz razloga što se svi podaci koji se prikazuju korisniku moraju dohvatiti s poslužiteljskog sloja, prvi je korak izrada mrežnog dijela aplikacije kako bi se ostvarila veza s poslužiteljem. Za slanje upita prema API-ju i čitanje podataka iz odgovora korištena je Retrofit biblioteka. Kako bi se implementirao Retrofit, potrebno je napraviti `EventService` sučelje s metodama koje odgovaraju pristupnim točkama definiranim u API-ju, a koje će se koristiti za slanje zahtjeva za resursima [10]. U kodu (Kod 4.7) nalaze se neke od metoda iz `EventService` sučelja koje prikazuju način definiranja metoda kod Retrofita.

```
    @POST("Event")
    suspend fun createEvent(@Body requestBody: RequestBody):
Response<Event>

    @PUT("Event")
    suspend fun updateEvent(@Body requestBody: RequestBody):
Response<Event>

    @GET("Event/UpcomingEvents")
    suspend fun getUpcomingEvents(): ArrayList<Event>

    @GET("Event/ForType/{eventType}")
    suspend fun getUpcomingEventsForType(@Path("eventType")
eventType: String): ArrayList<Event>
```

Kod 4.7 Metode EventService sučelja za objekte tipa Event

Kako bi se znalo koja HTTP metoda mora biti korištena pri zahtjevu, iznad metode se postavlja anotacija koja odgovara metodi koja se želi koristiti. Unutar te anotacije postavlja se i putanja do resursa kojem se želi pristupiti, što je uz HTTP metodu sastavni dio API zahtjeva [10].

API pozivi ne smiju preopteretiti memoriju, uzrokovati curenje podataka i uzrokovati pad aplikacije (engl. *crash*) ako dođe do greške. Zbog toga moraju biti asinkroni da bi radili ispravno za što se koriste korutine, odnosno `Coroutine`. Korutine su oblikovni obrazac

koji se koristi u Kotlinu za pojednostavljenje koda koji se izvršava asinkrono. Pomoću njih upravlja se dugotrajnim zadacima koji bi inače mogli usporiti aplikaciju i uzrokovati prestanak reakcije korisničkog sučelja [11]. Korutine u Kotlinu su slične `async – await` pristupu koji se koristi u C# programskom jeziku zbog toga što oboje služe za asinkrono izvršavanje određenog dijela programskog koda. Metode koje se izvršavaju `Coroutineom` koriste ključnu riječ `suspend`, što je slično korištenju ključne riječi `async` kod definiranja asinkronih metoda u C# jeziku.

Uz sučelje, potrebno je napraviti i `Network` klasu za kreiranje Retrofit klijenta kojom se pristupa metodama iz `EventService` sučelja.

Za krajnje korištenje metoda, odnosno poziva, implementirana je ViewModel klasa po imenu `EventViewModel`. U kontekstu razvoja Android aplikacija, ViewModel je klasa koja pruža podatke koji će se prikazivati na korisničkom sučelju, što omogućava odvajanje poslovne i prezentacijske logike aplikacije. Velika prednost kod korištenja ViewModela je čuvanje stanja, a to znači da se prilikom navigacije između aktivnosti korisničkog sučelja ili rotacije zaslona mobilnog telefona ne moraju svaki put dohvaćat podaci.

```
val eventsList = MutableLiveData<ArrayList<Event>>()

    fun getUpcomingEvents() {
        viewModelScope.launch {
            eventsList.value =
            Network().getService().getUpcomingEvents()
        }
    }
```

Kod 4.8 Dohvaćanje nadolazećih događaja u EventViewModel klasi

U kodu (Kod 4.8) prikazana je metoda za dohvaćanje nadolazećih događaja iz `EventViewModel` klase. U tijelu metode poziva se prethodno kreirana `Network` klasa putem koje se pristupa metodi iz sučelja. S obzirom na to da poziv vraća klijentu podatke, oni se spremaju u `eventsList` listu. Na identičan način su napravljene sve metode unutar `EventViewModel` klase.

S uspješno implementiranim mehanizmom za dohvaćanje podataka s poslužitelja kreće se na implementaciju poslovne logike svih aktivnosti korisničkog sučelja Prva aktivnost s kojom se korisnik susreće pri pokretanju aplikacije je `SignInActivity`. Korisnik se

može prijaviti u svoj korisnički račun unosom email-a i lozinke, a autentifikacija se odvija putem Firebase tehnologije. Klikom na gumb „Sign in“ okida se Firebase metoda koja traži korisnika na svojoj bazi podataka koji odgovara unesenom email-u i lozinki. S obzirom na to da ona vraća stanje uspješnosti autentifikacije, toj metodi se dodaje i slušatelj (engl. *listener*) koji prilikom završetka metode provjerava je li autentifikacija uspjela. U slučaju uspjeha, korisniku se otvara `MapsActivity` aktivnost i sve ostale funkcionalnosti aplikacije, a u slučaju da autentifikacija nije uspješna, prikazuje se poruka o neuspjeloj prijavi.

Ako korisnik želi napraviti korisnički račun, klikom na gumb „Register“ na aktivnosti `SignInActivity` prelazi na `RegisterActivity` aktivnost. U njoj popunjava formular s detaljima novog računa te klikom na gumb „Create account“; ako su sva polja točno popunjena, kreira novi korisnički račun. Zbog korištenja Firebase sustava, novi račun se kreira pomoću Firebase metode `createUserWithEmailAndPassword`. U kodu (Kod 4.9) je prikazano korištenje te metode koja, kao i metoda za prijavu, vraća stanje uspješne ili neuspješne registracije. Nakon uspješne registracije, korisnika se vraća na `SignInActivity` gdje se može prijaviti svojim novim računom.

```
mAuth.createUserWithEmailAndPassword(email,
password).addOnCompleteListener {
            if (it.isSuccessful) {
                addUserToDatabase(email, password)
                Toast.makeText(applicationContext,
"Registration successful!", Toast.LENGTH_LONG).show()
                val intent = Intent(this,
SignInActivity::class.java).apply {
                    putExtra(REGISTER_INTENT, 1)
                }
                startActivity(intent)
            } else {
                    Toast.makeText(applicationContext,
      "Registration unsuccessful!", Toast.LENGTH_LONG).show()
            }
        }
```

Kod 4.9 Registracija novog korisnika putem Firebasea

Nakon uspješne prijave u aplikaciju, korisniku se otvara `MapsActivity` koja služi kao glavna aktivnost aplikacije. Kako bi se prikazala Google karta na aktivnosti, potrebno ju je

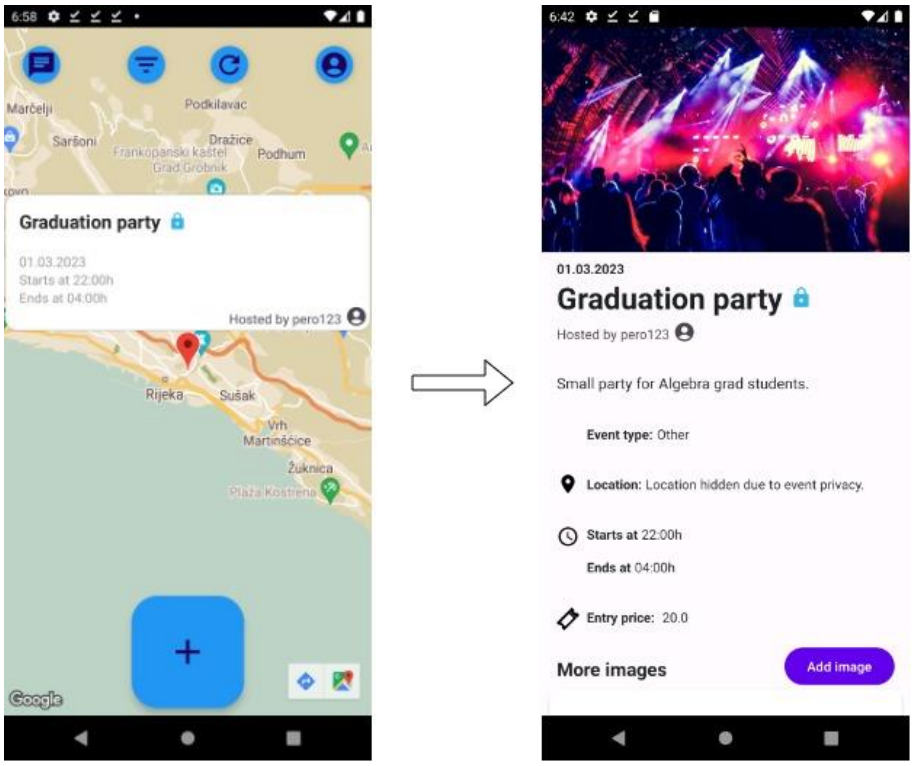
kreirati i pripremiti [12]. U `onCreate` metodu aktivnosti postavlja se uzvratni poziv koji postavlja kartu u korisničko sučelje kada je ona spremna za korištenje. Kako bi se kartu pripremilo i napunilo događajima za prikaz, koristi se ugrađena metoda `onMapReady`, prikazana u kodu (Kod 4.10). U njenom tijelu postavlja se stil karte, dohvaćaju događaji putem prethodno kreirane `EventViewModel` klase i postavljaju markeri na temelju lokacija događaja. Također, postavlja se i početna lokacija na karti.

```
override fun onMapReady(googleMap: GoogleMap) {
        mMap = googleMap

mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this,
R.raw.google_map_style))

mMap.setInfoWindowAdapter(EventInfoWindowAdapter(this))

        viewModel.eventsList.observe(this) { eventList ->
binding.progressBar.visibility = ProgressBar.VISIBLE

            eventList.forEach {
      val coordinates = it.locationCoordinates.split(',')
                val latLng =
LatLng(coordinates[0].toDouble(), coordinates[1].toDouble())
                val marker =
            mMap.addMarker(MarkerOptions().position(latLng)
                .title(it.title))
                marker!!.tag = it
            }
            binding.progressBar.visibility = ProgressBar.GONE
        }
        val rijeka = LatLng(45.339117, 14.402303)

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(rijeka,
12.0f))
        mMap.setOnMarkerClickListener(this)
        mMap.setOnInfoWindowClickListener(this)

        viewModel.getUpcomingEvents()
```

```
}
```

Kod 4.10 Priprema karte i događaja za prikazivanje

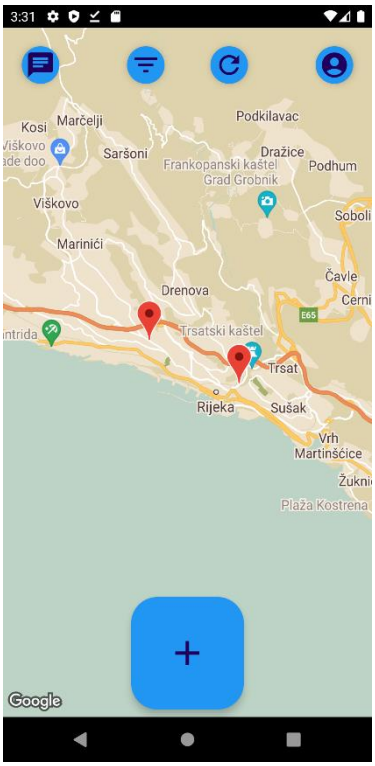
Klikom na marker nekog događaja, pojavljuje se prozor koji prikazuje nekoliko kratkih informacija o tom događaju, kao što su naziv, datum i vrijeme, privatnost događaja i organizator. Klikom na taj prozor otvara se `EventActivity` aktivnost sa svim detaljima odabranog događaja. U slučaju da je trenutno prijavljeni korisnik ujedno i organizator tog događaja, iz ove aktivnosti moguće je pristupiti `EventRequestActivity` aktivnosti ako je događaj postavljen kao privatan. U toj aktivnosti prikazani su zahtjevi drugih korisnika za prisustvovanje tom događaju koje organizator može prihvatiti ili odbiti. Popis zahtjeva prikazan je pomoću RecyclerView komponente, a za njenu implementaciju je potrebno kreirati klasu `RequestRecyclerAdapter` i tlocrt elementa popisa koji će služiti za prikaz svih elementa u popisu. `RequestRecyclerAdapter` se brine za ispravno prikazivanje pojedinačnih elementa popisa i informacija u njima, a kao parametar u konstruktoru prima listu objekata koji se žele prikazati. Uz to, nasljeđuje klasu `RecyclerView.Adapter` iz koje implementira tri metode: `onCreateViewHolder`, `onBindViewHolder` i `getItemCount`. `onCreateViewHolder` služi za kreiranje i prikaz elemenata, dok `onBindViewHolder` služi za prikaz informacija u svakom elementu [13]. `getItemCount` je jednostavna metoda koja vraća broj objekata iz liste. Na isti su način implementirane sve RecyclerView komponente u ovom radu, s razlikom u listi objekata koja se prikazuje i tlocrtu elemenata. Ako prijavljeni korisnik nije organizator događaja, prikazan je gumb za slanje zahtjeva. Na slici (Slika 4.4) prikazan je marker određenog događaja te `EventActivity` aktivnost za taj događaj.



Slika 4.5 Info prozor događaja i njegov EventActivity

`MapsActivity` na sebi sadrži i pet gumba za navigaciju koji korisnika vode do sljedećih funkcionalnosti:

- Filtriranje događaja
- Osvježavanje karte
- Popis razgovora
- Pregled detalja korisničkog računa
- Kreiranje novog događaja



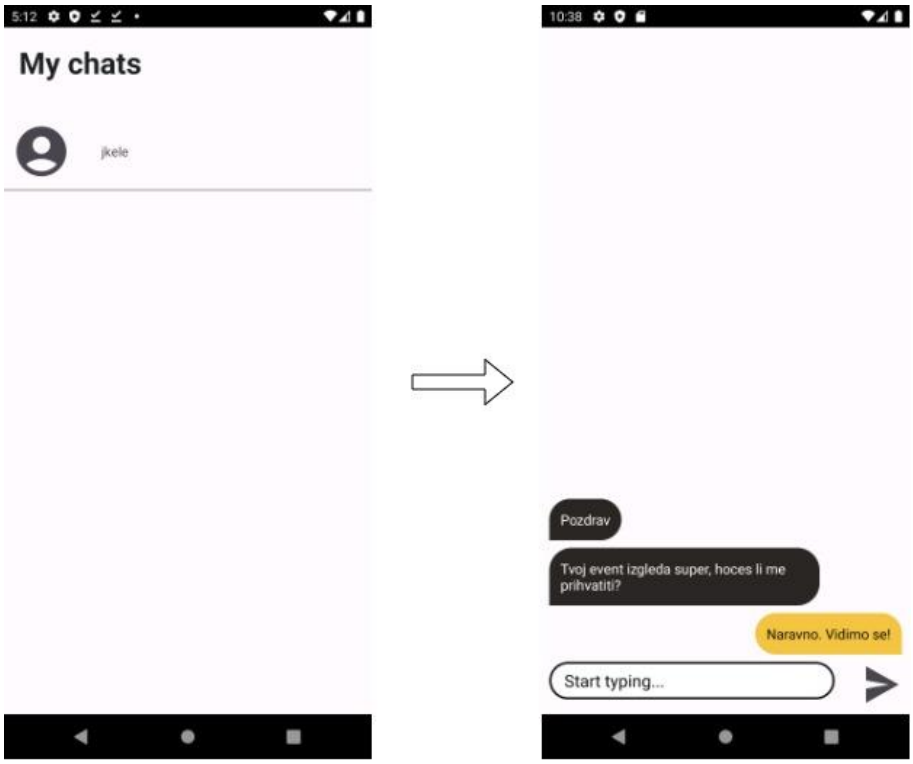
Slika 4.6 MapsActivity s kartom i navigacijskim gumbima

Klikom na gumb za filtriranje događaja otvara se prozor pomoću kojeg korisnik može odabrati tip događaja. Nakon odabranog tipa, na karti se prikazuju samo događaji koji odgovaraju tom tipu.

Osvježavanje karte radi na način da se ponovno dohvaćaju događaji s poslužitelja i postavljaju na kartu.

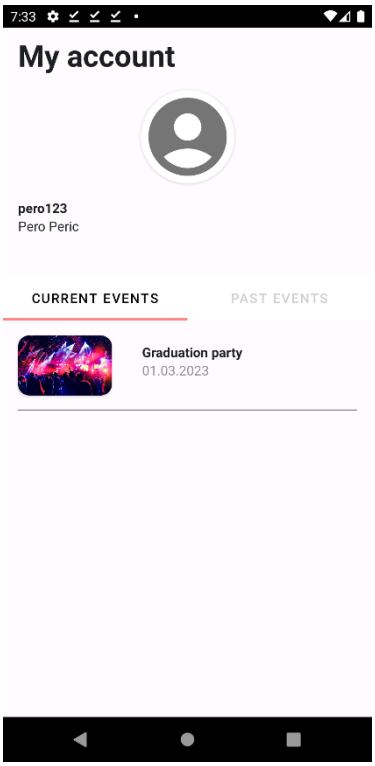
Gumb za popis razgovora vodi korisnika do `MyChatsActivity` aktivnosti koja prikazuje popis svih korisnikovih razgovora, a koju je moguće vidjeti na slici (Slika 4.6). Popis razgovora prikazan je pomoću RecyclerView komponente, a za njenu implementaciju je potrebno kreirati klasu `ContactRecyclerAdapter` i tlocrt elementa popisa koji će služiti za prikaz svih elementa u popisu. `ContactRecyclerAdapter` se brine za ispravno prikazivanje pojedinačnih elementa popisa i informacija u njima, a kao parametar u konstruktoru prima listu objekata koji se žele prikazati. Nasljeđuje klasu `RecyclerView.Adapter` iz koje implementira tri metode: onCreateViewHolder, `onBindViewHolder` i `getItemCount`. `onCreateViewHolder` služi za kreiranje i prikaz elemenata, dok `onBindViewHolder` služi za prikaz informacija u svakom

elementu. `getItemCount` je jednostavna metoda koja vraća broj objekata iz liste. Na isti način su implementirane sve RecyclerView komponente u ovom radu, s razlikom u listi objekata koja se prikazuje i tlocrtu elemenata.



Slika 4.7 Prozor popisa razgovora i prozor za razgovor

Za pregled detalja korisničkog računa, klikom na odgovarajući gumb otvara se `AccountActivity` aktivnost, prikazana na slici (Slika 4.7).



Slika 4.8 AccountActivity s detaljima korisničkog računa

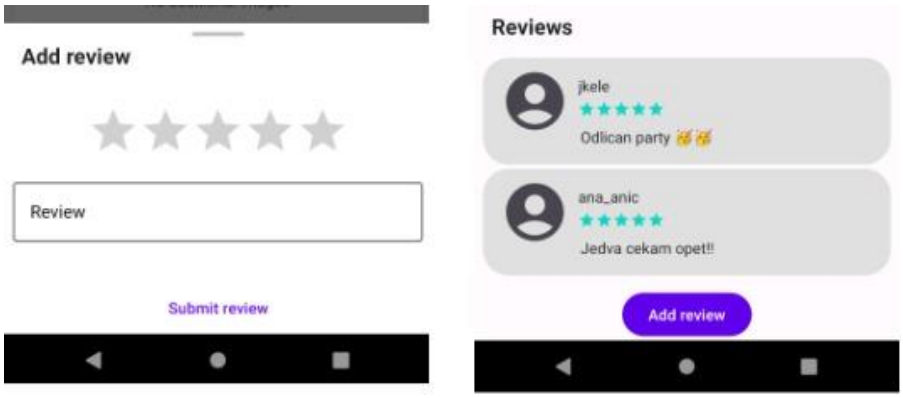
Detalji računa dohvaćaju se s API-ja preko AccountViewModel klase, zadužene za sve pozive prema API-ju koji rade s User objektom. U ovoj aktivnosti korištena su i dva fragmenta koja se izmjenjuju pomoću TabLayout tlocrtne komponente. Za njenu implementaciju potrebno je kreirati posebnu Adapter klasu koja se brine za prikaz fragmenata. U kodu (Kod 4.11) prikazana je implementacija navedene Adapter klase.

```
class AccountViewPagerAdapter(fragmentManager:
FragmentManager, lifecycle: Lifecycle, val userId: Int):
    FragmentStateAdapter(fragmentManager, lifecycle){
    override fun getItemCount(): Int {
        return 2
    }
    override fun createFragment(position: Int): Fragment {
        when(position) {
            0 -> return AccountCurrentEventsFragment(userId)
            1 -> return AccountPastEventsFragment(userId)
        }
        return AccountCurrentEventsFragment(userId)
    }
}
```

Kod 4.11 Adapter klasa za izmjenjivanje fragmenata

U prvom fragmentu nalazi se lista trenutnih događaja, dok se u drugom fragmentu prikazuje lista prošlih događaja korisnika. Obje liste se prikazuju pomoću RecyclerView komponente, a podaci za popunjavanje listi dohvaćaju se s EventViewModel klasom. Klikom na neki element iz liste otvara se EventActivity s detaljima odabrane aktivnosti.

Ako se radi od prošlom događaju, prikazuju se i recenzije za taj događaj pa je svaki korisnik u mogućnosti dodati novu recenziju. Za dodavanje recenzije, klikom na gumb „Add review“ prikazuje se prozor u kojem korisnik može dodati ocjenu od jedne do pet zvjezdica i kratku recenziju. Te informacije prikazuju se ostalim korisnicima u listi recenzija prikazanoj u RecyclerView komponenti. Na slici (Slika 4.8) prikazan je prozor dodavanja nove recenzije te izgled popisa recenzija.



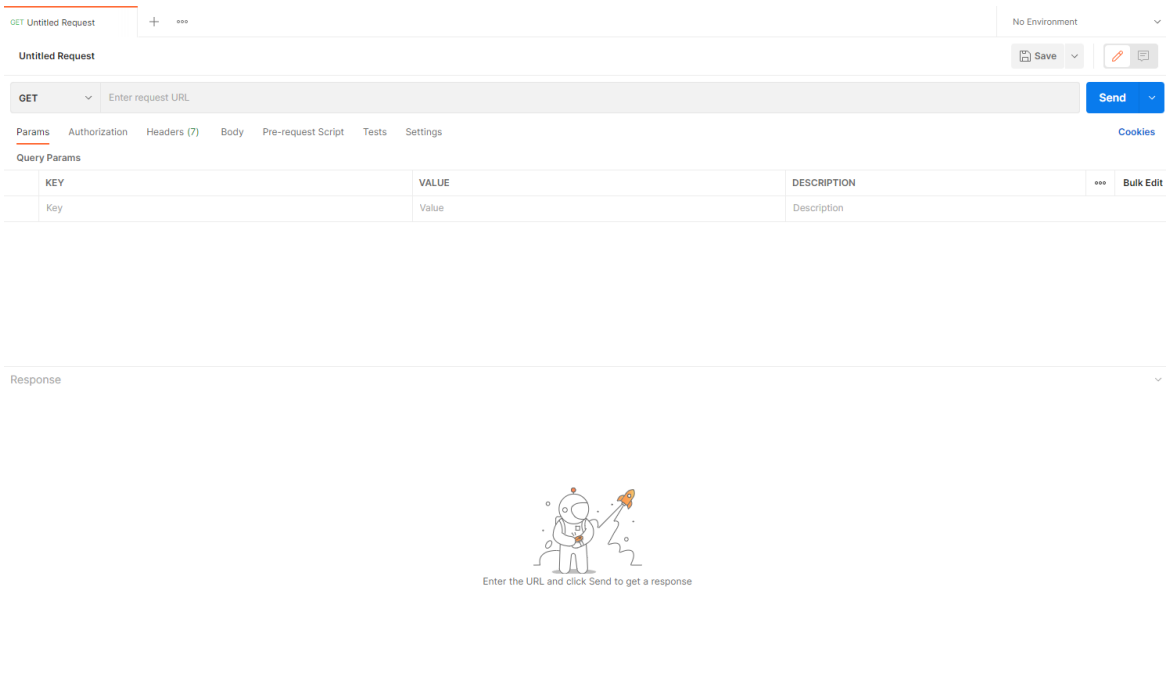
Slika 4.9 Prozor za dodavanje nove recenzije i prikaz recenzija za događaj

Kreiranje novog događaja moguće je ostvariti odlaskom na CreateActivity aktivnost, koja je prikazana na slici. Popunjavanjem formulara i klikom na gumb „Create“ kreira se novi događaj koji se zatim prikazuje svim korisnicima na karti. Prije stvaranja novog događaja, provjerava se jesu li unesene sve potrebne informacije. U slučaju da neke informacije nisu unesene ili su krivo upisane, korisniku se obraća pozornost na polja koja treba prepraviti.

5. Testiranje i analiza gotovog programskog rješenja

Gotovo programsko rješenje, koje je nastalo kao rezultat ovog rada, potrebno je testirati. S obzirom na to da se ono sastoji od više komponenti, zasebno će se testirati poslužiteljski i klijentski slojevi programskog rješenja.

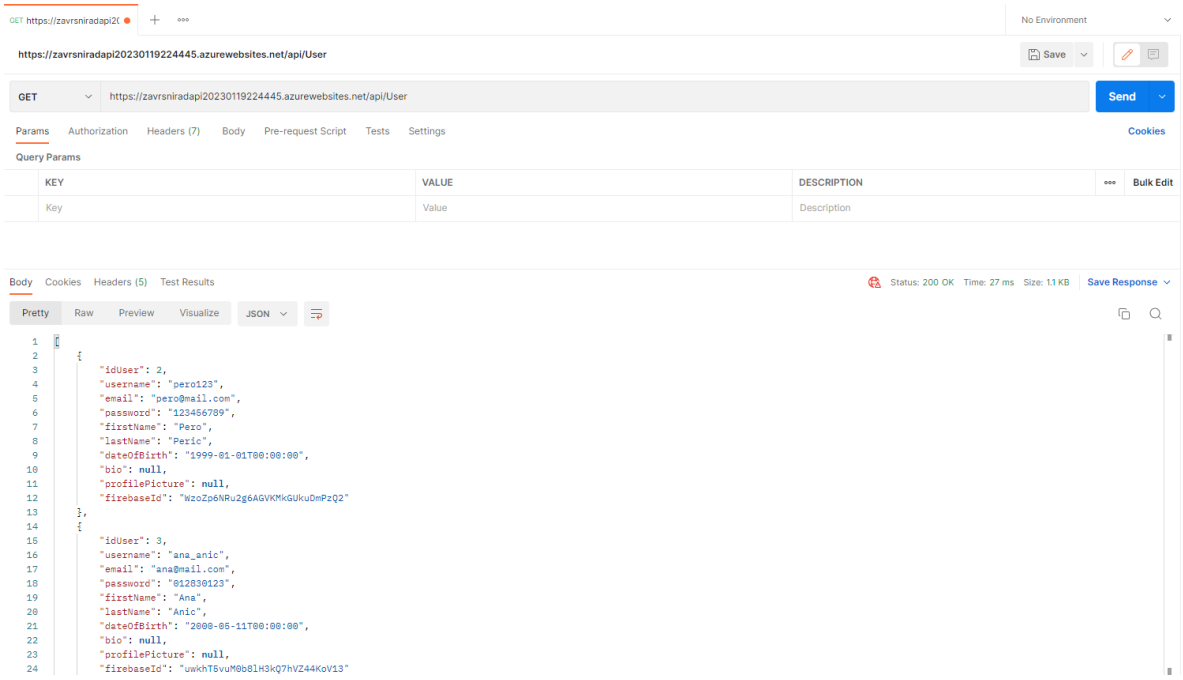
Kako bi se testirao poslužiteljski sloj, odnosno API, potrebno je provjeriti rade li sve pristupne točke koje su na njemu definirane. U slučaju da pristupna točka vraća neke podatke, provjerava se i točnost tih podataka. Za testiranje poslužiteljskog sloja korištena je Postman API platforma za izradu i korištenje API-ja [14]. Na slici (Slika 5.1) je prikazano radno okruženje Postman platforme koje je razdvojeno na dva dijela, dio za zahtjev prema poslužitelju (engl. *request*) i dio za odgovor koji se dobiva od poslužitelja (engl. *response*).



Slika 5.1 Radno okruženje Postman platforme

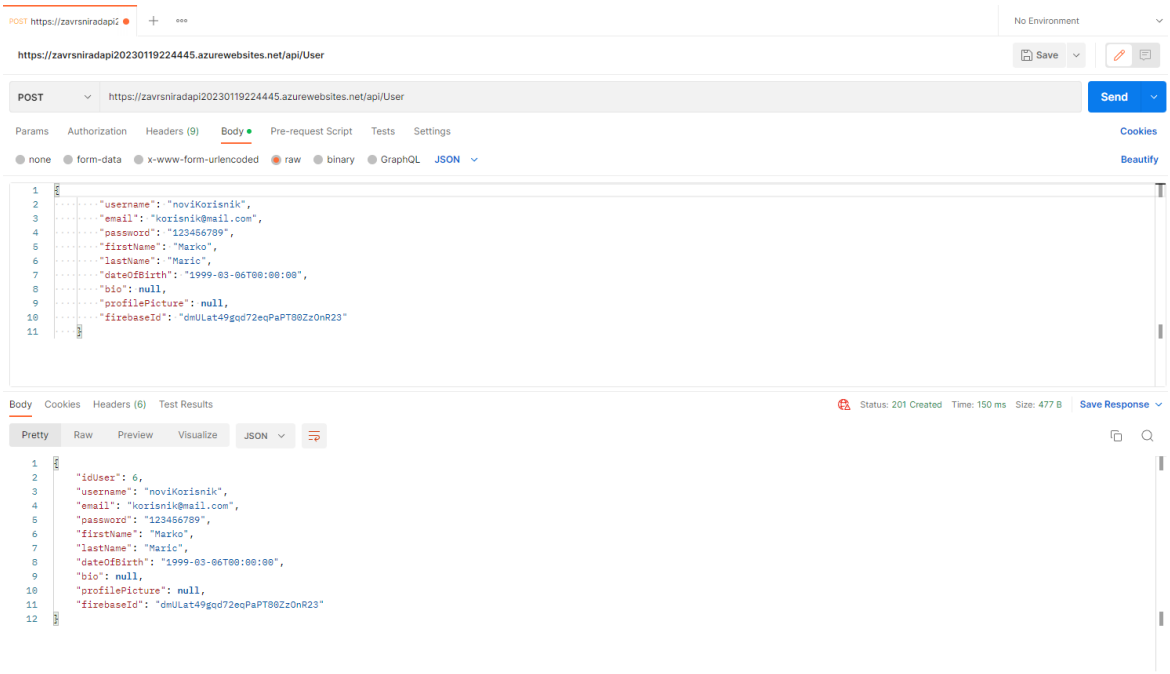
Dio za zahtjev sastoji se od polja u kojem je potrebno napisati URL zahtjeva te izbornika HTTP metode pomoću koje će se zahtjev izvršiti. Ovdje se upisuju svi zahtjevi koji se testiraju, jedan po jedan, te se odabire njihova odgovarajuća HTTP metoda. Klikom na gumb „Send“ zahtjev se šalje poslužitelju koji ga obrađuje i zatim šalje odgovor. Odgovor se prikazuje u JSON formatu u dijelu za odgovore, a uz njega su tu statusni kod zahtjeva i

vrijeme koje je bilo potrebno da se izvrši zahtjev. Na slici (Slika 5.2) može se vidjeti primjer testiranja zahtjeva za dohvaćanje popisa svih korisnika. Ako se zahtjev izvrši bez greške te podaci koji su vraćeni u odgovoru odgovaraju očekivanima, test se smatra uspješnim. Ovim načinom testirane su sve pristupne točke koje koriste GET metodu s jedinom razlikom u URL-u koji se koristi.



Slika 5.2 Testiranje zahtjeva za dohvaćanje popisa korisnika u Postman radnom okruženju

Zahtjevu je moguće postaviti parametre i zaglavlja u obliku ključ-vrijednost ako je to potrebno. Također, moguće je postaviti i tijelo zahtjevu, što se koristi kod zahtjeva za dodavanje novih ili ažuriranje postojećih objekata. Kako bi se testirali takvi zahtjevi, potrebno je promijeniti HTTP metodu u POST ili PUT, ovisno o vrsti zahtjeva. Zatim, promijeniti URL zahtjeva i na kraju u tijelo zahtjeva upisati objekt koji se želi stvoriti u JSON formatu. Klikom na gumb „Send“ zahtjev se izvršava. Uspješni zahtjev kao odgovor vraća novostvoreni objekt prikazanim u JSON formatu. Na ovaj način testirane su sve pristupne točke poslužitelja koje koriste POST i PUT metode. Na slici (Slika 5.3) prikazan je primjer testiranja zahtjeva za dodavanje novog korisnika.



Slika 5.3 Testiranje zahtjeva za dodavanje novog korisnika

Testiranje klijentskog sloja programskog rješenja provedeno je u testnom okruženju gdje se kroz dva testna scenarija prolazi kroz sve funkcionalnosti mobilne aplikacije. Prvi scenarij je korištenje aplikacije iz perspektive organizatora, gdje se testiraju funkcionalnosti stvaranja novih događaja i prihvaćanja ili odbijanja zahtjeva drugih korisnika.

Drugi scenarij testiranja je korištenje aplikacije iz perspektive običnog korisnika, odnosno korisnika koji samo pretražuje događaje. Iz ovoga gledišta, testiraju se funkcionalnosti pretraživanja događaja na karti, što uključuje filtriranje događaja. Pregledavaju se i prozori s detaljima pojedinačnog događaja kako bi se utvrdilo da prikazuju sve informacije na točan način. Testira se i funkcionalnost slanja zahtjeva za privatne događaje, gdje se i provjerava stanje zahtjeva nakon prihvaćanja ili odbijanja od strane organizatora. Uz to, testira se i dodavanje recenzija za završene događaje.

Funkcionalnosti izrade novoga korisničkog računa, prijave u korisnički račun, razgovora s drugim korisnicima i pregledavanje detalja korisničkog računa testira se u oba scenarija s obzirom na to da su to funkcionalnosti koje se dotiču svih korisnika, bez obzira na način korištenja aplikacije.

Izradom i testiranjem ovoga programskog rješenja proizlazi prototip sustava za upravljanje događajima koji je spreman za korištenje. Prije početka izrade rješenja postavio se njegov

glavni cilj, a to je olakšavanje pronalaska i promoviranja događaja manjeg opsega. S obzirom na to da je klijentski dio sustava ona komponenta kojom će krajnji korisnici zapravo rukovati, implementacija njenih funkcionalnosti i dizajn korisničkog sučelja su provedeni tako da se osigura ostvarenje glavnog cilja. Same funkcionalnosti raspoređene su po aplikaciji na način da korisnik u nekoliko klikova može doći do onog što želi. Tijekom navigacije kroz aplikaciju, korisnika se ne zagušuje nepotrebnim i dodatnim sadržajem, kao što je to prisutno kod nekih postojećih rješenja.

Gotovo rješenje ima određene prednosti i nedostatke koji se mogu promatrati iz perspektive pojedinog sloja sustava, ali i sustava kao cjeline. Promatrajući sustav kao cjelinu, glavni nedostatak je korištenje Microsoft Azure servisa za objavu podatkovnog i poslužiteljskog sloja zbog čega pate performanse sustava. Iz perspektive poslužiteljskog sloja, najveći nedostatak je manjak sigurnosnih mjera za korištenje API servisa. Zbog nedostatka sigurnosnih mjera, ovom servisu može pristupiti bilo tko, što znači da podaci kojima se pristupa nisu u potpunosti sigurni.

Uzimajući u obzir da je gotovo programsko rješenje prototip, ono ima prostora za napredak i nadogradnju, a prvi bi korak bio pronalazak boljeg rješenja za objavu baze podataka i poslužitelja kako bi se poboljšale performanse sustava. Moguća su rješenja izrada vlastitog servera ili korištenje postojećih usluga, a nakon odabira optimalnog rješenja, sljedeći bi korak bio migracija baze i poslužitelja na odabrano rješenje. Za poboljšanje sigurnosti podataka, kojima se pristupa putem poslužitelja, implementiralo bi se korištenje sigurnosnih ključeva. Time se zahtjevi prema poslužitelju izvršavaju samo ako sadrže odgovarajući sigurnosni ključ.

Kako bi se podržali svi pametni mobilni telefoni, razvila bi se i mobilna aplikacija za iOS uređaje. Uz iOS mobilnu aplikaciju, postoji mogućnost i izrade web-aplikacije kako bi se sustav mogao koristiti i na osobnim računalima putem internetskog preglednika.

Također, postoje i dodatne funkcionalnosti koje bi se implementirale na klijentskom sloju kako bi se unaprijedilo korištenje sustava. Primjer takve funkcionalnosti bila bi funkcionalnost pomoću koje korisnici mogu izraziti zanimanje za neki događaj kako bi organizatori, ali i ostali sudionici, znali koliko osoba mogu očekivati na svom događaju. Također, unaprijedila bi se postojeća funkcionalnost filtriranja događaja pomoću dodavanja novih filtra za poboljšanje pretrage događaja.

Zaključak

Organizacija manjih događaja i sudjelovanje u njima sastavni su dio socijalnog života ljudi. Na početku ovog rada postavljen je cilj koji je olakšavanje pretrage i promoviranja događaja manjeg opsega kako bi se korisnicima pojednostavio taj proces, što na kraju rezultira uštedom vremena.

Analizom tržišta prikazana su trenutno postojeća rješenja koja koristi velik broj ljudi diljem svijeta. Detaljnijom analizom tih rješenja prikazane su sve prednosti i nedostatci koje ona sadrže. Prikazani nedostaci su motivacija za ovaj rad iz razloga što se razvojem novog sustava, koji nastaje kao rezultat rada, pokušava pronaći rješenje za navedene nedostatke.

Uz analizu trenutnih rješenja za pronalazak i promociju takvih događaja, u sklopu rada provedeno je i istraživanje kako bi se utvrdila potreba za sustavom fokusiranim isključivo na manje događaje, a koji nastaje kao rezultat ovog rada. Istraživanje je dokazalo da je većina ispitanika nezadovoljna postojećim rješenjima i da bi nova mobilna aplikacija olakšala njihovu potragu i promociju manjih događaja.

Kroz rad je opisana arhitektura programskog rješenja i sve njegove komponente uz definiranje uloga pojedinačnih komponenti. Zbog velike količine tehnologija koje se koriste za realizaciju programskog rješenja, u sklopu komponenti opisane su i objašnjene sve tehnologije korištene prilikom razvoja ovog rješenja.

Najveći je dio rada posvećen klijentskom sloju sustava, odnosno Android mobilnoj aplikaciji, gdje se opisuju funkcionalnosti aplikacije i pojedinačne komponente korištene za realizaciju korisničkog sučelja. Uz opis funkcionalnosti, prikazuje se i proces razvoja i implementacije funkcionalnosti na razini koda.

Na kraju, da bi se osiguralo da programsko rješenje i sve njegove komponente rade kako je zamišljeno, provedeni su testovi koji provjeravaju rad poslužiteljskog i klijentskog sloja sustava. Uz testiranje, provedena je analiza gotovoga programskog rješenja koja prikazuje rezultat ovog rada, njegove prednosti i nedostatke te mogućnosti napretka i nadogradnje u budućnosti.

Popis kratica

API	*Application Programming Interface*	aplikacijsko programsko sučelje
CRUD	*Create, Read, Update, Delete*	stvoriti, čitati, ažurirati, izbrisati
HTTP	*Hypertext Transfer Protocol*	protokol prijenosa hiperteksta
JSON	*JavaScript Object Notation*	JavaScript objektna notacija
MSSQL	*Microsoft SQL Server*	Microsoft SQL server
ORM	*Object Relational Mapping*	objektno relacijsko mapiranje
REST	*Representational State Transfer*	prijenos reprezentativnog stanja
SDK	*Software Development Kit*	komplet za razvoj softvera
SQL	*Structured Query Language*	jezik strukturiranih upita
URI	*Uniform Resource Identifier*	uniformni identifikator sadržaja
URL	*Uniform Resource Locator*	ujednačeni lokator sadržaja
XML	*Extensible Markup Language*	proširivi označni jezik

Popis slika

Slika 2.1 Rezultat prvog pitanja....7

Slika 2.2 Rezultat drugog pitanja....8

Slika 2.3 Rezultat trećeg pitanja....9

Slika 2.4 Rezultat četvrtog pitanja....9

Slika 2.5 Rezultat petog pitanja....10

Slika 2.6 Rezultat šestog pitanja....10

Slika 2.7 Rezultati sedmog i osmog pitanja....11

Slika 3.1 Skica arhitekture sustava....12

Slika 3.2 Skica *Code-first* pristupa pri korištenju Entity Frameworka....15

Slika 4.1 Dijagram odnosa entiteta....20

Slika 4.2 Prozor za objavu web-servisa....27

Slika 4.3 Verzije Android operacijskog sustava s postocima distribucije....28

Slika 4.4 Info prozor događaja i njegov EventActivity....36

Slika 4.5 MapsActivity s kartom i navigacijskim gumbima....37

Slika 4.6 Prozor popisa razgovora i prozor za razgovor....38

Slika 4.7 AccountActivity s detaljima korisničkog računa....39

Slika 4.8 Prozor za dodavanje nove recenzije i prikaz recenzija za događaj....40

Slika 5.1 Radno okruženje Postman platforme....41

Slika 5.2 Testiranje zahtjeva za dohvaćanje popisa korisnika u Postman radnom okruženju....42

Slika 5.3 Testiranje zahtjeva za dodavanje novog korisnika....43

Popis kôdova

Kod 4.1 AppDbContext klasa 21

Kod 4.2 IEventRepository sučelje 22

Kod 4.3 AddEvent metoda za kreiranje novog događaja 23

Kod 4.4 GetEvents metoda za dohvaćanje događaja 25

Kod 4.5 Metoda GetEventsForType 25

Kod 4.6 Kod za dodavanje Firebase biblioteka 28

Kod 4.7 Metode EventService sučelja za objekte tipa Event 32

Kod 4.8 Dohvaćanje nadolazećih događaja u EventViewModel klasi 33

Kod 4.9 Registracija novog korisnika putem Firebasea 34

Kod 4.10 Priprema karte i događaja za prikazivanje 35

Kod 4.11 Adapter klasa za izmjenjivanje fragmenata 39

Literatura

[1] *Number of monthly active Facebook users worldwide* (2022) Dostupno na: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/ [14. siječnja 2023.]

[2] FIELDING, T.R., NOTTINGHAM, M., RESCHKE, J. (2022) *HTTP Semantics* Dostupno na: https://httpwg.org/specs/rfc9110.html#overview.of.status.codes [15. siječnja 2023.]

[3] GILLIS, A.S. *REST API (RESTful API)* (2020). Dostupno na: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API [15.siječnja 2023.]

[4] *Getting Started with EF Core* (2022) Dostupno na: https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli [16. siječnja 2023.]

[5] *Kotlin Programming Language* (2023) Dostupno na: https://kotlinlang.org/ [17. siječnja 2023.]

[6] *Firebase* (2023). Dostupno na: https://firebase.google.com/solutions [18. siječnja 2023.]

[7] *Material Design for Android* (2022) Dostupno na: https://developer.android.com/develop/ui/views/theming/look-and-feel?hl=en [19. siječnja 2023.]

[8] *C# guide – await operator* (2023) Dostupno na: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await [15. veljače 2023.]

[9] PATTANKAR, M., HURBUNS, M. *Mastering ASP.NET Web API.* Birmingham: Packt Publishing, (2017), 89-100

[10] *Retrofit* Dostupno na: https://square.github.io/retrofit/ [15.veljače 2023.]

[11] *Kotlin coroutines on Android* (2022) Dostupno na: https://developer.android.com/kotlin/coroutines [15.veljače 2023.]

[12] *Maps SDK for Android* (2023) Dostupno na: https://developers.google.com/maps/documentation/android-sdk/overview [15. veljače 2023.]

[13] HORTON, J. *Android Programming with Kotlin for Begginers*. Birmingham: Packt Publishing, (2019), 393-405

[14] *What is Postman?*. Dostupno na: https://www.postman.com/product/what-is-postman/ [17. veljače 2023]